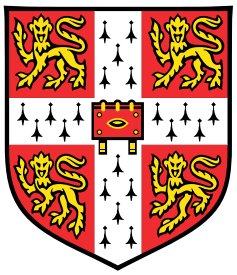
Supervisory Control of a Multi-Arm Telerobotic System Supported by Mixed Reality



Clare Heinbaugh

Department of Engineering
University of Cambridge

This dissertation is submitted for the degree of
Master of Philosophy

Clare College August 2024

I would like to dedicate this dissertation to my family, Margaret, Andrew, Charles, and Eleanor, friends both in the United States and United Kingdom, and my two cute dogs, Shark Bait and Tofu.

Declaration

I, Clare of Clare College, being a candidate for the MPhil in Machine Learning and Machine Intelligence, hereby declare that this report and the work described in it are my own work, unaided except as may be specified below, and that the report does not contain material that has already been used to any substantial extent for a comparable purpose.

I wrote all of the code to support this work using Python, C#, C++, and Dart, except for the gaze-tracking calibration script written by John Dudley, the original STANet model architecture script written by Matt Song, and the two device haptic feedback script provided by Touch X. I used Google Cloud Platform to store my hand gesture dataset and deploy my annotation tool.

The word count of this document including tables, footnotes, figure captions and appendices is 13083 words.

Clare Heinbaugh
August 2024

Acknowledgements

I would like to thank the Marshall Scholarship for financially supporting my Master's degree and easing my transition to the UK. This work would not be possible without the support of my advisors John Dudley, Arsen Abdulali, and Matt Song, so thank you for your time and efforts. I also want to thank my study group, Sedie Simpson, Elijah Gutierrez, Andrej Jovanović, and Darius Feher, for your constant support, friendship, and willingness to participate in my user studies.

Abstract

Currently, remote control, or teleoperation, of multiple robots requires a lot of time and energy from operators. Existing methods to combat this issue use various combinations of cameras, three-dimensional scene renderings, and methods of control (Hokayem and Spong, 2006; Pace et al., 2021; Van de Merwe et al., 2019; Warattaseth et al., 2024). Control of a multi-arm setup is limited by the focus and precision of an operator, i.e., it is harder for operators to control multiple arms at the same time. Warattaseth et al. (2024) demonstrated that using abstracted motion primitives instead of direct control to actuate a robot arm reduced active control time by 75%. Active control time refers to the time spent by the operator indicating which actions the robot should perform. To leverage this efficiency, I extrapolate to multi-arm control and demonstrate robot arm actuation on a pick-and-place task. In a novel approach to this problem, I use a virtual reality (VR) headset to capture hand and gaze input and an attention-based model to recognize gestures. I show that users are able to more accurately control multiple arms simultaneously, compared to direct control with a haptic input device. Furthermore, the multi-modal input (both hand- and gaze-tracking) reduces the sequence of selecting an action and then an object to perform that action on to a parallel process.

Table of contents

List of figures

List of tables

Chapter 1

The problem with the claw machine arcade game

1.1 Problem context

Teleoperation is the act of controlling one or more robots from a remote location (Hokayem and Spong, 2006). Teleoperation and human-robot collaboration is often used in manufacturing and healthcare (Peppoloni et al., 2015; Sinha et al., 2015). For instance, in gynecology, the Da Vinci surgery system is widely used for operations and consists of four robot arms: three to perform surgery and one to control the camera (Sinha et al., 2015). Currently in the setup described by Sinha et al. (2015), a 2D input system is used to control the robots, allowing doctors to manipulate only one arm at a time. In general, effectively operating multiple robot arms at the same time is slower and more challenging.

Warattaseth et al. (2024) proposed one possible solution to accelerate and ease task completion. Warattaseth et al. (2024) actuated a single robot arm using the head-tracking feature of mixed reality (MR) headsets and abstracted motion primitives that they coined Control with Motion Primitives (CMP), communicated via a haptic input device and interpreted with a supervised machine learning (ML) model. In this paper, I refer to CMP as abstracted control. By operating at a higher abstraction level compared to direct control, i.e., as the user moves their hand, the robot copies directly, they were able to achieve a 75% reduction in active control time (Warattaseth et al., 2024). Active control time is determined by how long the user is conveying instructions to the robot. Furthermore, the perceived workload was reduced using the CMP method (Warattaseth et al., 2024).

Unfortunately, in a single-arm system, these savings only translated to free time as the users waited for the single robot arm to execute their instructions, minimally reducing

the overall task duration. In my work, I implement a multi-arm system, so that users can convey multiple instructions in short succession and two or more arms can operate in parallel. Besides reducing task completion time, an efficient multi-arm system supported by abstracted control could reduce surgery times, make it easier for surgeons to learn to operate, and improve precision.

Another change I make to the setup proposed by Warattaseth et al. (2024) is using the hand-tracking and eye-tracking capabilities of newer virtual reality (VR) headsets to leverage multi-modal inputs to tackle more complex tasks that require discernment between objects. For instance, I leverage eye-tracking to determine which object a user wants to interact with. Conversely, in the setup proposed by Warattaseth et al. (2024) there was only one object to grab, move, rotate, and drop, so discernment was not necessary.

By demonstrating a more efficient and natural method of teleoperation, abstracted control can be applied to a variety of applications for improved human control of multiple robots.

1.2 Direct control versus abstracted control

In the arcade claw game, users are asked to maneuver a robot arm, press a button, and lift up a prize. In practice, most complain that nabbing a prize is incredibly difficult and lament spending any money on the game at all. The claw game is an example of direct control of a robot arm: as a user adjusts the position of the joystick, the arm moves at a fixed speed in the corresponding direction. A more general direct control setup would be a robot hand tracking a user's hand exactly, e.g., moving in three-dimensional space with the user's hand and opening and closing a robot grabber in sync with the user.

In contrast, motion primitives are abstractions of movements a user performs via direct control. In a direct control scenario with the more advanced robot setup, a user may find it challenging to move their hand through space precisely enough to move an object from point A to point B. Using motion primitives, they can perform a gesture to indicate “move" and look at where they want to move to (point B). In a setup that can recognize hand movements and gaze, the system could infer that the user wants to move the object from its current position to the place where the user is looking. Through this abstracted control procedure, users can more accurately and quickly convey commands to a robot. We will refer to this method as “abstracted control."

1.3 Research questions

The four tenets to measure robot-human performance include:

- efficient instruction passing from human to robot,
- efficient system status passing from robot to human,
- a shallow learning curve for the user, and
- low cognitive load for the user.

as discussed by Wang et al. (2019). These tenants will inform the research questions that I aim to address with a user study comparing abstracted and direct control on a pick-and-place task. I use a think-aloud study to gather qualitative feedback on the proposed system design.

The questions I address through this work are:

Q1 Can abstracted control convey user intent more efficiently to the robots compared to direct control?

Q2 Can abstracted control reduce the mental strain of teleoperating multiple robots?

Q3 Are gaze selection and hand gestures, in particular, a straightforward method of abstracted control for users?

1.4 Approach

In this work, I describe a system composed of a VR application that tracks hands and gaze. This data is streamed in real time to a server that predicts which gesture the user is performing on a particular objects and sends instructions to physical robot arms. An external camera is used to update the scene to the server and VR application in near real-time. This set up removes the burden on the operator by allowing them to express intent implicitly through gaze and gestures. It also allows users to exploit the freedom that accompanies extracting out low-level robot control.

1.5 Road map

This work proceeds with a synthesis of related literature relevant to the method of abstracted control introduced in this chapter. Afterwards, I describe the full system design, key components, and flow of information. Then, I focus on just the gesture classifier, including the data collection process and model training. Next, I evaluate the system through a user study and end with a discussion of limitations and future work.

Chapter 2

Related works

2.1 Overview

In this chapter, I examine previous research related to the areas of robot teleoperation and human-robot collaboration, gesture-based control, and using gaze-tracking for specifying objects. These are the major areas related to my research: investigating how multi-modal inputs at a higher level of abstraction can ease and accelerate robotic teleoperation.

2.2 Robot teleoperation and human-robot collaboration

Teleoperation is the act of a human commanding a robot, without being physically present at the scene (Hokayem and Spong, 2006). Regardless of the remote environment, humans need input from the physical setup to make operating choices, whether that input be as simple as raw floats representing positions or an exact replica of the real-time environment rendered in three dimensions using VR. Pace et al. (2021) conducted a user study on the effectiveness of teleoperation in VR, in particular assessing robot arm control with tasks often focused on actuating the robot end effector (joint farthest from the robot base). In particular, Pace et al. (2021) found that visualization of the entire robot arm, not just the end effector, could aid task performance.

Up until the last few years, research has been mixed about the importance of providing context in a VR scene for effective teleoperation (Burdea, 1999; Van de Merwe et al., 2019). The literature refers to this as telepresence (Van de Merwe et al., 2019). VR can provide high telepresence because the scenes can be to scale and offer depth information, which are advantages over a traditional camera feed (Wonsick and Padir, 2020). Van de Merwe et al. (2019) compared a fully rendered version of the remote environment and a preprocessed,

simplified version of the remote environment (only rendering objects necessary for task completion) and found that participants completed a robot line-following task quicker and with more reported ease. Furthermore, Tung et al. (2021) implement a pick-and-place task (which they refer to as “multi-cube lifting") to demonstrate their work in multi-arm control via teleoperation. Thus, in my teleoperation task where depth perception and scale matter, I use a VR headset and to-scale three-dimensional scene to create high telepresence for users in a similar pick-and-place task.

2.3 Hand-tracking and gesture-based control

Human gesturing has been documented in literature for over half a century (Karam and m. c. schraefel, 2005). Karam and m. c. schraefel (2005) create a gesture taxonomy to review gesture work across the field of human-computer interaction (HCI), including highlighting VR as an application of gesture-based interactions. Like LI et al. (2019), they conclude that gestures should feel “natural" and “intuitive" to users (Karam and m. c. schraefel, 2005). Song et al. (2023) take this a step further showing how gesture-based interactions in VR that are metaphoric in functionality, e.g., rotating one finger corresponds to rotating an object, can help users recall gestures easier. Drawing on these broad classification and previous works that use gestures and technology to indicate user intent, I define a series of gestures that are also metaphoric in nature. The main question then is how to translate these hand movements into output that can be used to train a machine learning model. Thus, we consider various methods of hand-tracking.

Each hand gesture is a continuous stream of joint positions over time. Due to technological limitations, we discretize this continuous stream into frames observed or inferred from some input device. Previous research has relied on physical controllers and haptic input devices to approximate or abstract hand positions composing hand gestures (Abdlkarim et al., 2023; Almeida et al., 2017; Warattaseth et al., 2024). Warattaseth et al. (2024) continuously streamed the absolute positions resulting from forces applied to the Touch X haptic input device to create “hand gestures." For instance, the “placing" gesture was composed of the position, rotation, and button states as the user moved the control stick downwards, recording at 1,000 Hz (Warattaseth et al., 2024). Warattaseth et al. (2024) draw upon previous work using haptic devices, including Ly et al. (2021) who used haptic input devices to predict user intent in a teleoperation scenario. However, all of these methods rely on an external input devices that force particular hand shapes, e.g., gripping motion to hold onto the Touch X control stick, limiting the possible hand postures.

Abdlkarim et al. (2023) showed that the newer VR devices like the Meta Quest 2 can accurately perform “markerless" hand-tracking. “Markerless" technology uses a series of machine learning-based methods to track a user’s hand in a VR application. First, a segmentation step parses the hands from the background (Abdlkarim et al., 2023). Then, the joint positions are identified (Abdlkarim et al., 2023). Finally, the joint positions are followed through time in a tracking stage (Abdlkarim et al., 2023). Like the Meta Quest 2, the Meta Quest Pro, which I use in this reseach, outputs hand-tracking data: joint positions and rotations labeled to allow tracking through time.

Even though hand gestures may feel more natural to users (Karam and m. c. schraefel, 2005; LI et al., 2019), a study by Masurovsky et al. (2020) found that a controller-based setup produced higher performance and usability both qualitatively and quantitatively compared to a physical setup. Despite this result, they suggest that technological or modelling advances in machine learning could enable an easier free hand experience. In fact, in the years since Masurovsky et al. (2020) compared a custom Leap Motion vision-based gesture recognizer to an Oculus Touch controller solution, newer setups leveraging Quest VR devices have been able to accurately track hands and interpret gestures (Abdlkarim et al., 2023).

2.4 Gaze-tracking for specifying object to act upon

For many HCI tasks, there are two distinct steps:

1. specifying to what object/where the system should apply the action, and
2. requesting the system to perform an action (Song et al., 2023).

For instance, in a pick-and-place task, the operator must first specify *which object* that they want to pick up and then indicate that they want to *pick up* the object. Similarly, if a user wants to place the object in a particular location, they must specify *the location* the object should be placed and the *action of placing* the object. Recent works have aimed to simplify, or even combine these two steps. For example, Song et al. (2023) developed HotGestures. In their work, the location in VR space where a user wants to apply an action is also where they perform that action (Song et al., 2023). For instance, if the user wants to draw a three-dimensional sphere, they make a fist in the place that they want the sphere to appear (Song et al., 2023).

Similarly, in my work, I allow the two steps described above to be performed in parallel rather than in sequence. I choose to leverage multi-modal input: hand-tracking to specify the action and gaze-tracking to specify what object or position the action affects. This allows users to complete the two step process quicker.

2.5 What sets this work apart

In the system design that I propose in the next chapter, I use gestures and gaze-selection to reduce the procedure of conveying using intent from a sequential to a parallel process. This method of abstracted control allows users to actuate multiple robot arms at the same time and aims to reduce the overall task completion time initially hypothesized by Warattaseth et al. (2024). The system is evaluated qualitatively via a user study and shows promising improvements in terms of ease of use and efficiency over direct control.

Chapter 3

System design

3.1 Overview

I will now develop the full system design. This chapter is structured to motivate the final design, starting with the basics of controlling the robots. Next, I describe how the abstracted and direct control conditions can be used to control the robots. Finally, I detail the auxiliary components needed to support these mechanisms of control.

More specifically, first, I will describe how I use a game loop to provide parametric movement updates to the robot arm. For the direct control case, I will explain how I transform the tracked point of the haptic input devices to the movement of the robot arm end effector, while applying force bounds for safe operation. For the abstracted control case, I will explain how I use hand gestures to specify an action and gaze-tracking to specify which object the action should affect. Finally, I will explain how an external camera is used in both cases to track the interactable objects and positions; interactable objects are blocks that the user can select with gaze and interactable positions are the places the user can select to place blocks.

Figure 3.1 shows the final setup and highlights the components that will be enumerated and connected in this chapter including the:

- robot arms,
- haptic input devices,
- gesture classifier,
- external camera,
- interactable objects and positions, and

- gaze-tracker.

Figure 3.2 describes the system design at a high-level for the abstracted control case, showing how information flows between the major components: user, VR application, server, external camera, robot controller, physical robot, and the simulated robot. Figure 3.3 shows the diagram of the direct control case, composed of the user, VR application, haptic input devices, server, external camera, robot controller, physical robot, and simulated robot. Notably, I replace input from the user's free hands with haptic device positions and omit the inference step to determine which gesture the user is performing. I also remove the gaze-tracking in the direct control case.

All data flows through a central Python server that handles computations such as coordinate system transformations, model inference, and conditional robot control. The server is built using FastAPI and is running locally on a Windows laptop with a graphics processing unit (GPU) that can be accessed by other devices on the same network via its IPv4 address.

3.2 Basic robotics control

Extending from control of one robot arm to two robot arms necessitates parallel actuation of both arms. I drew on the Unity game loop setup. In Unity, every component has a `Start` function, called once on setup, and a `Update` function, which gets called every time the scene re-renders[1]. On server startup, I instantiate robot controllers for both the left and right arms, and then use a linear parametric function to compute the update to the robot end effector. The end effector is the tip of joint farthest from the base of the robot, as shown in Figure 3.4.

The robot controller holds the following values: target position and rotation, and current position and rotation[2]. For context, our target position could be set in the full pipeline by either execution of one of the gestures or the haptic input devices. I use a parametric linear model to break the path between the current and target positions into a series of points that the robot moves to on each call to the robot position update function, just like the Unity game loop.

Let $\mathbf{x_c} = [x_c, y_c, z_c]$ be the current position and $\mathbf{x_t} = [x_t, y_t, z_t]$ be the target position. Define,

$$\Delta \mathbf{x} = \mathbf{x_t} - \mathbf{x_c}$$

[1]The Quest Pro has a maximum refresh rate of 90 Hz or once every 0.011 seconds.

[2]For smooth movement of the arm, I only query the receiver of the physical robot arm on startup, to prevent rapid adjustments to the positions that occur if we use the exact position as the current position instead of an internally maintained value. The shaking occurs because of the delay between requesting the state and receiving the state and adjusting the linear parametric model due to small fluctuations in the position.

and

$$\widehat{\Delta \mathbf{x}} = \frac{\Delta \mathbf{x}}{||\Delta \mathbf{x}||}$$

$\widehat{\Delta \mathbf{x}}$ is a unit vector pointing from our current position $\mathbf{x_c}$ to the target position x_t. Thus, the new current position $\mathbf{x_c}$ after elapsed time Δt, given constant speed s, is shown in Equation 3.1.

$$\mathbf{x_{c'}} = \mathbf{x_c} + \widehat{\Delta \mathbf{x}} \cdot \Delta t \cdot s \tag{3.1}$$

The calculation for the rotation update mirrors the position update, except that I allow for a different speed s', to account for scaling differences between updates to the position and rotation. In my final implementation, I compute the distance to the target position and rotation given the current position and rotation, and if either exceeds a threshold α, I leverage Equation 3.1 to update the current position. If the offset does not exceed α for either position or rotation, I set the current position and rotation to the target rotation and position, to minimize any numerical errors causing slight offsets.

The physical robot arm also has a built in proportional-integral-derivative (PID) controller, which applies the necessary forces to counter gravity to hold the robot in the target position. Thus, once the arm reaches the destination position, the robot arm receives no additional position or rotation updates but maintains the target position exactly. In contrast, the simulated robot setup that we generate using Pybullet does not have a built-in PID controller, so we need to continuously send position updates. With the dual physical and simulated robot set up, we can use the same robot controller and swap in either the physical or simulated robot arms. In either case, I use the built-in inverse kinematics solver to compute joint positions from three-dimensional position updates[3] Figure 3.5 shows the communication between the server, robotic controller, physical robotic arm, and simulated robotic arm. The “grabber" is an electromagnet. When switch on, the end effector attachment is magnetized and can pick up other magnets. When switched off, magnetic objects are released.

3.3 Direct control with haptic input devices

Warattaseth et al. (2024) leverage haptic input devices to allow both direct control and gesture-based control of a single robot arm. Similarly, Ly et al. (2021) also use haptic input

[3]There are multiple solutions to this problem. This became an issue when running with the simulated model, where one of the controllable joints would make contact with the simulated table in the setup, because of the inverse kinematic solution it found. To remedy this, we start both the physical and simulated models with the controllable joints as far from the table as possible, and the kinematics solutions chosen stay away from the table during testing.

devices to infer user intent for teleoperation. Following their work, I use two haptic input devices for control of the left and right robot arms in the direct control case. I use a websocket to send the position and rotation data of the haptic device to the web server to be further communicated to the robot arms. To ensure that the movement of the robot arm is fixed despite the speed that the user adjusts the device positions, I leverage the robot controller logic, adjusting the target position and rotation only, and incrementally moving the arms at a fixed rate on each update step. Furthermore, I apply a force proportional and opposite to the force applied by the user outside of pre-determined bounds as follows:

```
forceVec[0] = (left_robot_bounds[0] - pos_1[0]) * force_multiplier;
```

This barrier-like force prevents the user from moving one arm too close to the other one[4].

3.4 Abstracted control with model to recognize gestures

Figure 3.6 shows the structure breakdown of the hand data sent from the VR app to the server. I use the `which_hand` attribute to extract the hand performing the movement and add the `SkeletonData` object to the corresponding hand buffer. Once one of the buffers (either left or right hand) has n number of objects, I restructure the `BoneData` object into a tensor, concatenating the position (`Vector3`) and `Quaternion`. The tensor is passed through a gesture classifier and a gesture is output. The gesture classifier is a supervised machine learning model. The data collection for and training of this model is described in detail in Chapter 4.

In Chapter 4, I train a model on six gestures: grab, move, drop, rotate, roll, and flip. I also consider non-gestures, called “null gestures," which are the hand movements (or non-movements) performed by users between the main gestures. Despite training on six main gestures and one null gesture class, we consider only the grab and drop gestures in the user study described in Chapter 5. All other gesture predictions are re-classified as null gestures. I establish the robotic control look-up table shown in Figure 3.7 for grab and drop.

3.5 Tracking interactable objects and positions

I have two types of physical objects that users can use the robot arms to interact with: interactable objects and interactable positions. In the user study described in the Chapter 5, the task is to place interactable objects, in this case cubes, on interactable positions, locations

[4]For additional safety, I also wrote a server-side validation function leveraged by the robot controller that only sets target positions within a pre-specified cube.

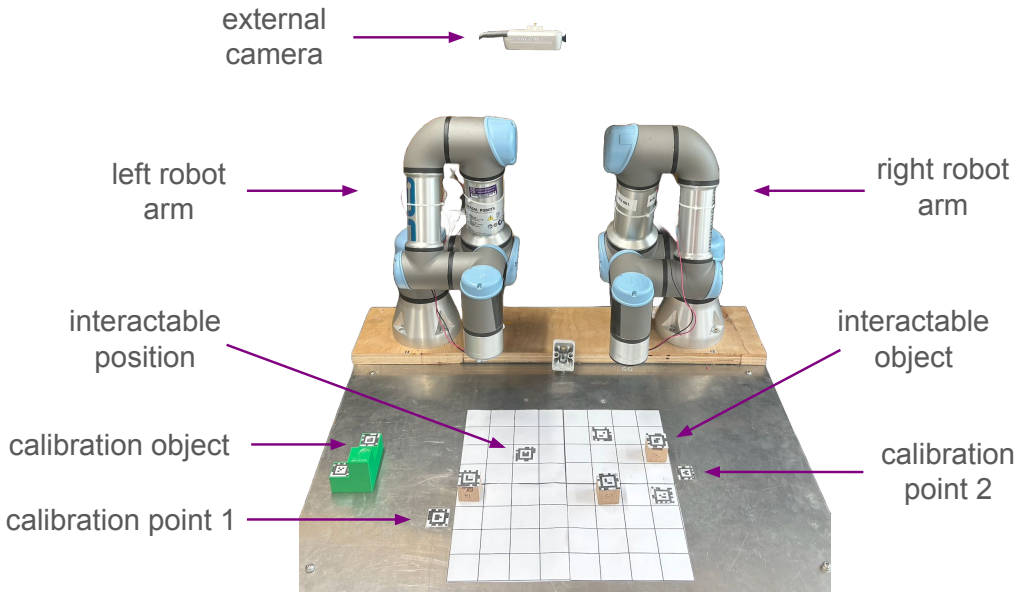
marked on the table that the robots are positioned above. In order to relay the locations of these interactable objects and positions in real-time, I installed an Intel Real Sense camera above the robot arms; the camera relays the scene as a two-dimensional surface with the table as the backdrop at a maximum rate of 30 frames per second. Each interactable object and position is marked with an April Tag (Olson, 2011). April tags were developed for tracking robots, and I am using the `tagStandard41h12` family of tags. Each tag has an associated ID, and `tagStandard41h12` has $2,115$ possible IDs.

In my setup, interactable objects are marked with tags corresponding to odd-numbered IDs, and interactable positions are marked with even number-bearing tags. Additionally, interactable objects correspond to interactable positions with a value one greater. For example, the interactable objects with ID 5 belongs on the interactable position with position 6. These pairings are used to render the objects in virtual reality and ultimately determine success on a particular task: all interactable objects need to be placed on their corresponding interactable position.

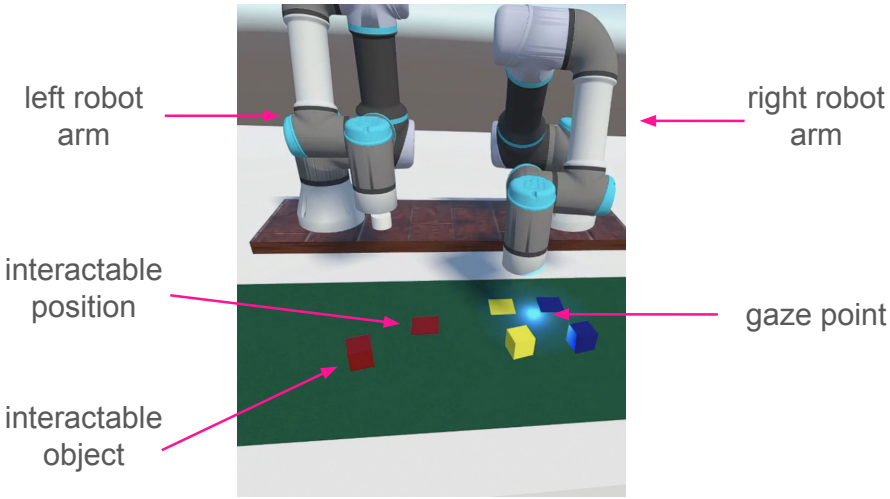
The external camera data is passed through a Python script that returns the (x,y) coordinates of each interactable object and interactable position. I then apply a linear transformation, defined by fixed calibration points, to convert the (x,y) positions in the external camera coordinate system to (x,y) coordinates in the reference frame of the left robot arm. The coordinate system is shown in Figure 3.8. All of my blocks in the final set up are the same height, so the z coordinate of each object is fixed and equal in the left arm coordinate system.

3.6 Gaze-tracking

The Meta Quest Pro has gaze-tracking technology. Using Unity, I am able to match the position and orientation of the user's eye and construct a ray that extends from their eye to objects in the VR scene to create physical intersections. I always consider the first intersection as the object of interest. Figure 3.9 shows the ray construction in a sample VR app. I average the position of the gaze from both eyes and stabilize the end position by considering an average over 10 gaze measurements (Dudley, 2024). Gaze-tracking is used for object/position selection in the final user study.



(a) This setup shows all components of the physical system.



(b) This setup shows all components of the simulated system.

Fig. 3.1 Real and virtual system setups.

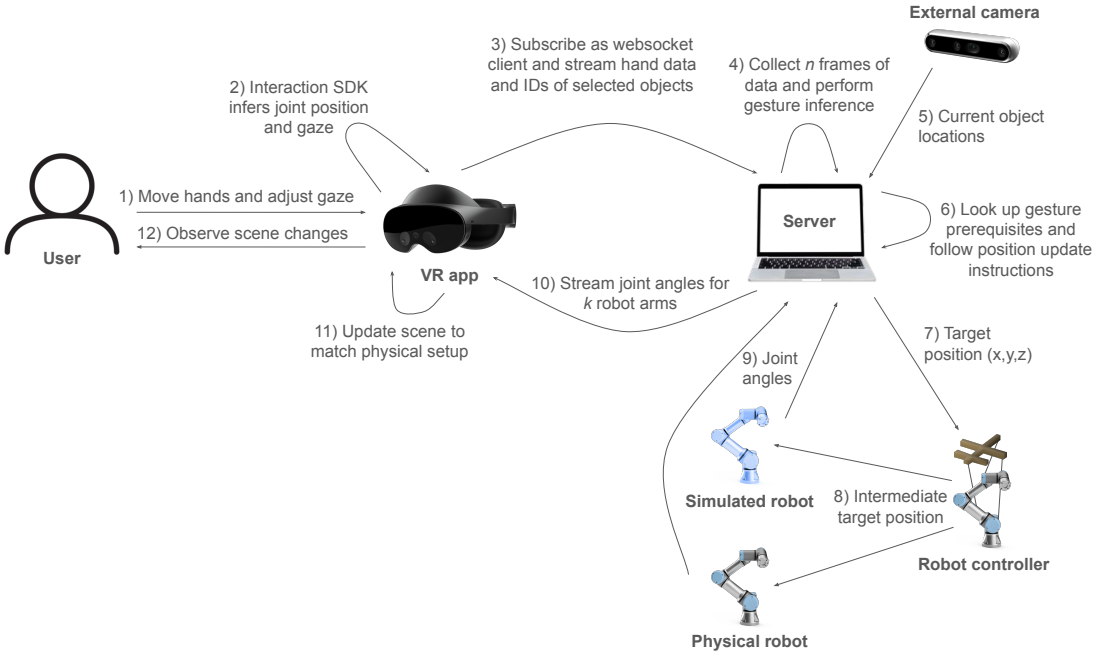


Fig. 3.2 Complete diagram for abstracted control case. I trace through flow of information at a high-level here. 1) The user moves their hands and adjusts their gaze. In the abstracted control case, hand gestures are used to request certain robot actions. Gaze is used to select objects and destination positions in the scene. 2) I leverage the built-in Interaction SDK to get the user's current joint position and gaze on every frame re-render, which is 3) passed via a websocket. Websockets allow for a continuous stream of data on every frame re-render. 4) The server receives the current hand position and the IDs of the selected objects. Using a buffer, the server waits for at least n frames of hand data for each individual hand before 6) performing gesture inference. 5) Meanwhile, an external camera sends the current coordinates of objects in the physical scene. Once a gesture has been observed and associated with an object to act on, 7) the robot controller accepts target positions and 8) instructs both the physical and simulated robots to take a step towards the new target position. 9) The server queries the k robot arms for joint angle updates and 10) streams these via a websocket back to the VR headset. Using this new data, 11) the headset updates the VR scene, and the 12) user can observe the changes and decide what to do in response.

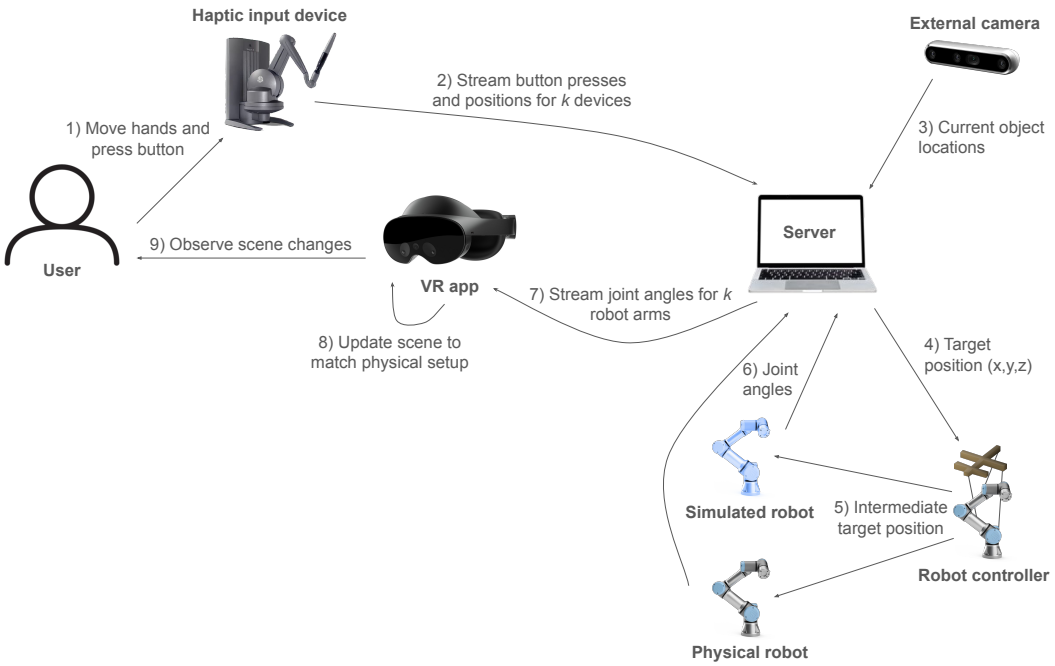


Fig. 3.3 Complete diagram for direct control case. 1) Users adjust the position of the k haptic input devices with their hands and can press the onboard buttons. 2) Whether or not the button was pressed and the current position of each of the k input devices is streamed directly to the server. Like the abstracted control case, the current position of the scene objects are 3) tracked via an external camera. 4) The new target position is set for each of the k robot arms, which is proportional to the input positions. Then, for smooth and safe movement, the 5) physical and simulated robot arms are set to an intermediate position, heading in the direction of the new target position. 6) The joint angles are communication to the server and then 7) to the VR app. 8) Finally, the scene is updated internally in the VR app and 9) made visible to the user wearing the headset.

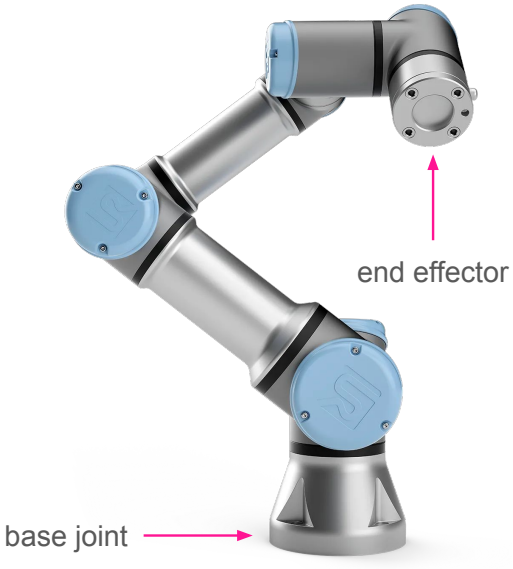


Fig. 3.4 Diagram of single robot arm. The base joint of the left arm is used as the universal origin reference point. The end effector is the controllable joint furthest from the base joint, indicated in this diagram. This is the point that I set via gaze tracking in the abstracted control case and via haptic input device position in the direct control case.

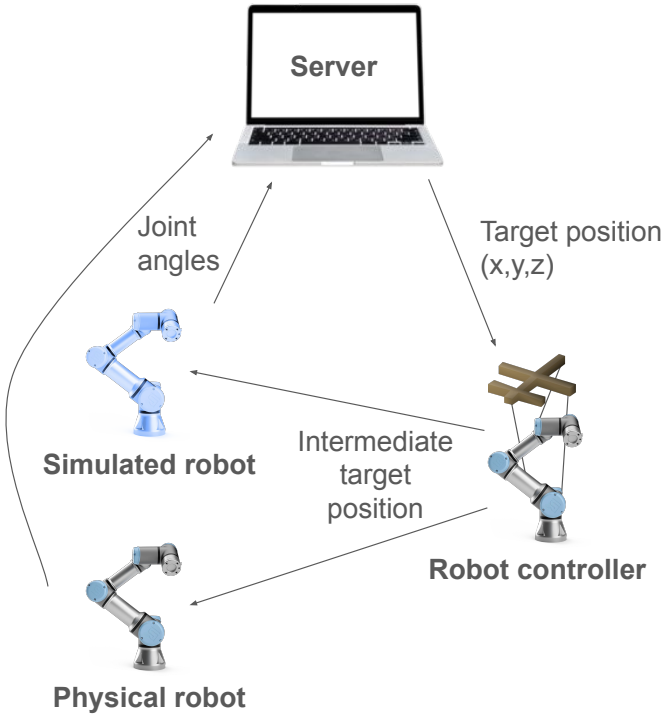


Fig. 3.5 Zooming in on one part of the full setup, we have the robot control diagram. Here, the target position is set via the server. The robot controller keeps track of the current position of the robot arm and sets the new target position. To avoid the robot arms jumping to this position, creating unsafe behavior especially in the direct control case, we get the elapsed time and compute an appropriate intermediate step towards the overall target position by setting the intermediate target positions. Then the robot controller computes an inverse kinematics solution to rotate the robot arm joints. The arm moves via a call to the `servoL` function and the new joint angles are sent back to the server. `servoL` is a function provided by the `ur_rtde` library. Unlike `moveL`, `servoL` does not set the starting and ending acceleration to 0. When used to update the robot arm many tiny steps at a time, using `servoL` leads to smooth transitions rather than sinusoidal accelerations and decelerations every step towards the target position.

```
class SkeletonData(BaseModel):
    bone_data: list[BoneData]
    which_hand: str

class BoneData(BaseModel):
    bone_id: int
    position: Vector3
    rotation: Quaternion
```

```
class Vector3(BaseModel):
    x: float
    y: float
    z: float

class Quaternion(BaseModel):
    x: float
    y: float
    z: float
    w: float
```

Fig. 3.6 A `SkeletonData` object is passed on every re-render of the Unity app via a web-socket to the server for processing. The `SkeletonData` object is composed of 24 bones each with 7 features, a concatenation of the `Vector3` values and `Quaternion` values.

Grab

Prerequisites
- Target interactable object must not be null.
- Position of target interactable object must be reachable by one of the arms.
- Electromagnet must be switched off for chosen arm.

Instructions
- Move arm to interactable object.
- Turn on electromagnet.
- Move back to reset position.

Drop

Prerequisites
- Target interactable position must not be null.
- Position of target interactable position must be reachable by one of the arms.
- Electromagnet must be switched on for chosen arm.

Instructions
- Move arm to interactable position.
- Turn off electromagnet.
- Move back to reset position.

Fig. 3.7 Grab and drop look-up table for robotics control. If all prerequisites are met for a particular gesture, then the robot controller executes the gesture instructions asynchronously to allow additional gesture calls to be received and process.

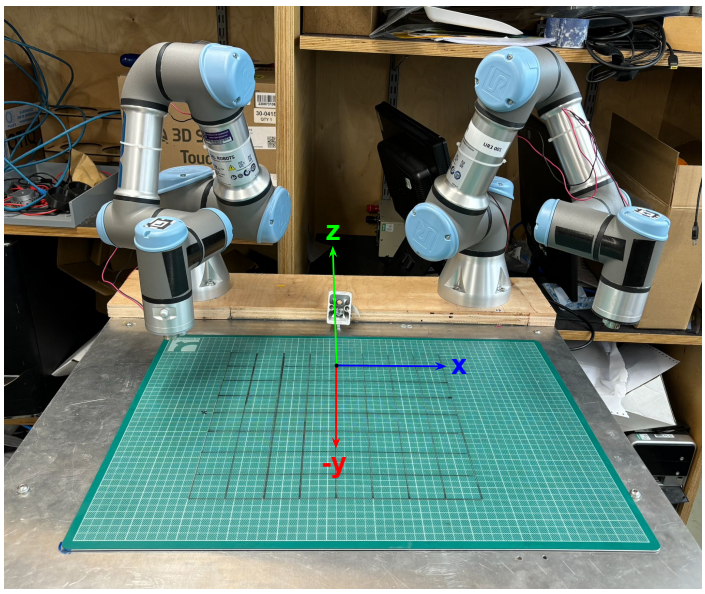


Fig. 3.8 Axes indicated on real setup. The universal origin is placed at the bottom of the base of the left robot arm, however for figure clarity, we show the axes in the middle of the setup.

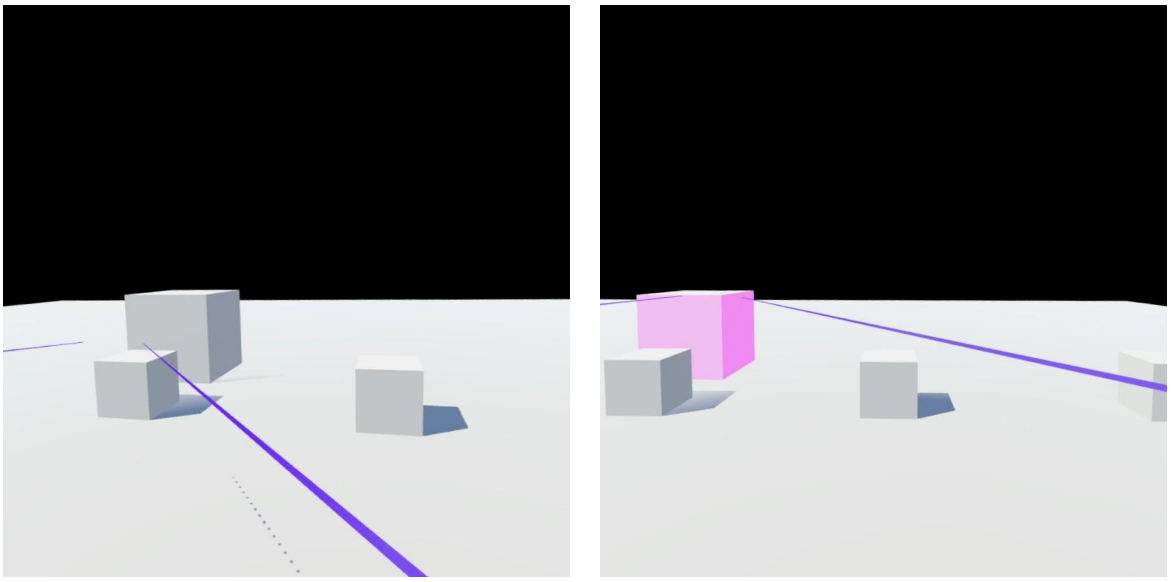


Fig. 3.9 Gaze-tracking demonstration Unity app. The purple rays are used to determine intersections between the user's eyes and objects in the scene that are part of the `interactable` layer. On the left we set the center of the two rays not intersecting with any interactable objects, and on the right we see the center intersecting with the back left cube, hence the visible feedback of a now pink cube. The gaze tracker always returns the first object of intersection. For instance, if the user was gazing slightly farther down, only the front left block would be returned.

Chapter 4

Hand gesture recognition

4.1 Overview

To perform online inference for abstracted control of a robotic system, we need a trained gesture classifier. To train this model, we need a dataset. I collect data as users perform various hand gestures. In this case, the input device is the Meta Quest Pro camera. In this chapter, I explain the gestures, including non-gestures called “null gestures," and how they are collected via a VR data collection app. Then, I discuss the short user study used to collect the data and the implementation for the custom web-based tool that I built to easily annotate the data.

After pre-processing the data collected to extract gestures, I consider various models to classify gestures. First, I find a baseline accuracy of always predicting the majority class (null gestures), with a simple, standard ML model. Next, I discuss transformers and how they can be applied to the problems of recognizing hand gestures in a state-of-the-art model called STANet. I present results from fine-tuning and show a confusion matrix and training curves for the best-performing, seeded model, which achieved 97.03% test accuracy. This model is leveraged in the abstracted control case to infer gestures in real-time. I descibe the process of online recognition including some of the challenges that led to model modifications including inputting relative positions and wrist velocity, separating the left and right hand models, and reducing the number of frames per sample.

4.2 Designing the gestures

While in the previous chapter, I only described a final setup for a pick-and-place task, I originally envisioned 3 tasks for multiple robot arms:

1. pick-and-place,
2. rolling out pizza dough, and
3. building a table.

These are expanded upon in Section 6.6.2 as part of Future Works, but they are necessary to introduce here.

After determining which tasks I wanted to accomplish with the setup in the final user study, I worked backwards and designed gestures to complete each sub-task. For the pick-and-place task, I created gestures to grab and drop. For the pizza dough rolling task, I considered gestures to grab, move, rotate (about the z-axis), drop, and roll the rolling pin on the dough. For the table construction task, I added the flip gesture for rotation around the x-axis. Figure 4.1 shows the hand shape and movement for all gestures.

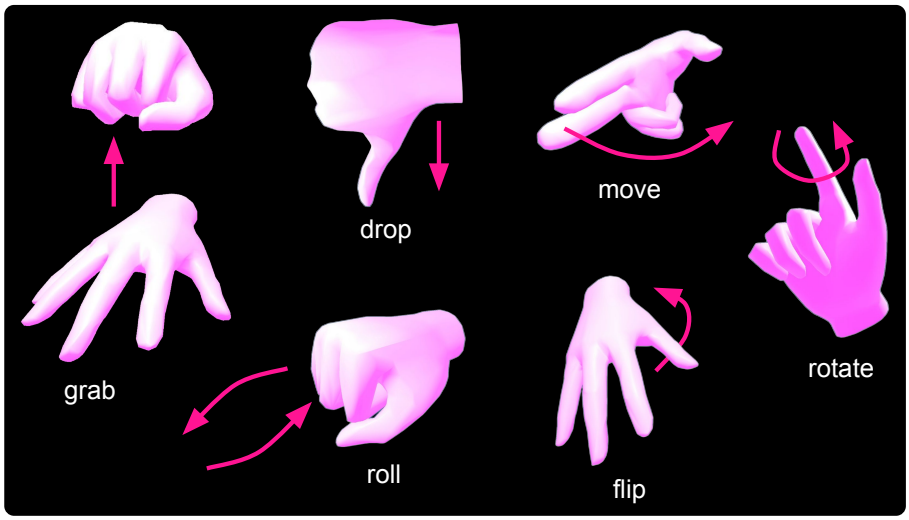


Fig. 4.1 Diagram of six main gestures: grab, drop, move, roll, flip, and rotate. All gestures are dynamic and the pink arrows show the direction of motion.

4.3 Null gestures

In addition to the six main gestures, I also needed to consider null gestures, the poses a user holds their hands in and the movements they perform when not performing a recognized gesture. Xu et al. (2022) had users wear an Apple watch and perform what they called “non-gestures," but I will call null gestures. The most relevant movement prompts to my research included asking participants to “tap[] on the watch/other surfaces" and “scratch[]

head/hands" (Xu et al., 2022). Any motions that they performed during this time composed the non-gesture class (Xu et al., 2022). As such, I decided to record three gestures in sequence during the data collection phase. The pauses before and after each gesture form the null gesture class. Finally, at the end of the data collection procedure, I asked users to do a random mix of motions with the hands including touching the VR headset, putting their hands in their lap, wiggling their fingers, looking at their hands, and counting to ten with their fingers. These additional ten-second recordings provide further null gesture data.

4.4 Hand gesture dataset collection application

One of the most popular benchmark datasets for gesture classification is the SHREC 2017 Track dataset. To curate for the dataset, researchers collected 14 hand gestures one to ten times using one finger or the entire hand, recorded with a RealSense depth camera Smedt et al. (2017). Following their setup, I designed six gestures, and asked participants to perform each gesture one to five times. The reasoning was to prevent fatigue from requiring too many trials.

Using Unity's animation tool, I created animations for each of the six gestures. After entering the user's participant ID and dominant hand, users were shown each of the six animations, performed by demonstration hands in the tutorial phase of the trial. They had the option to watch the six gesture demonstration as many times as they wanted, although all participants watched it exactly once, having practiced the gestures before putting on the VR headset. Then, participants began the recording phase. They were shown three gesture recordings in a row using the demonstration hands, watched a 3-2-1 countdown, and then were asked to repeat the gestures they just saw, with the gesture names appearing in the app. For additional visual feedback, the user's hands were represented by a blue VR Material when the data was not being collected and a green Material when recording.

I chose to include three gestures per trial because it was not too many to remember how to perform but enough to generate null gesture data between individual gestures. Between gestures, the demonstration hands were in a particular resting position, and participants were explicitly told that they did not need to match the demonstration hand positions between gestures. Participants had up to ten-seconds to perform the sequence of three gestures for a single trial. If they were unhappy with a particular recording, they had an option to verbally ask me to delete the recording. Upon completion of all three gestures before the ten-seconds had elapsed, users were asked to say "Done" and the recording was stopped early. After ten trials, the users were asked to record null gestures during a final 10-second trial. Figure 4.2 shows screenshots from the VR data collection app.

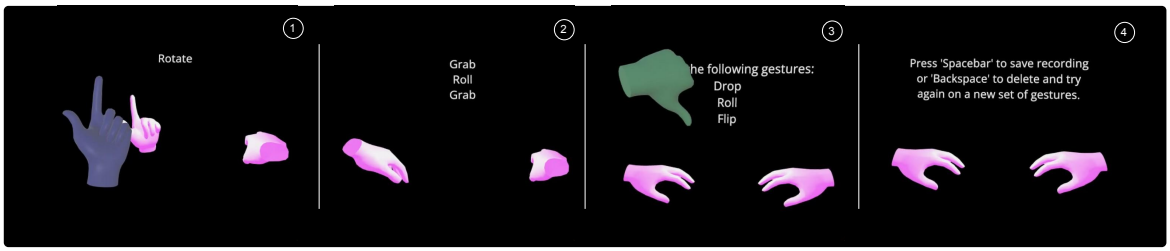


Fig. 4.2 Gesture recorder Unity app. 1) The app starts with a tutorial. The user's hands are in blue when hand data is not being recorded, and the pink hands demonstrate the gestures. 2) Next, the user is shown a sequence of 3 random gestures, both in writing and demonstrated by the pink hands. 3) Afterwards, the user sees a countdown from 3 and their hands turn green. They are instructed to perform, in order, the gestures listed on the screen. The demonstration, pink hands are still for this part. 4) After a recording is complete (i.e., the user says "Done" and I press stop or 10 seconds elapse), the user can either choose to delete the recording and try again on a new sequence of 3 random gestures or move on to the next random sequence.

The VR app connects to this a FastAPI websocket endpoint to stream hand-tracking data, collected using Meta's Interaction SDK package. Once a JSON file containing the pose information for a series of time steps is saved to a Google Cloud storage bucket, a second script converts this data into GIFs of both the left and right hands using the Python package Matplotlib. I then created a simple annotation tool using Flutter, shown in Figure 4.3, that loads the GIFs from the external database and shows them to the user along with the expected gestures. Using the slider at the bottom of the tool and gesture dropdown menu, I can easily select a start and end frame to label for a particular gesture. After reviewing the entire GIF and marking all gestures, I submit the data which is passed back to the external Google server and saved as a CSV file.

4.5 Dataset collection

Prior to participating the data collection process, all participants read a Participant Information Sheet explaining the study's purpose and general procedure, signed a User Consent Form, and filled out a Demographic Questionnaire. I then demonstrated all of the gestures to the user by saying the name of the gesture out loud and performing the motion with my own hands. After they could confidently remember and perform each gesture, users put on the VR headset and progressed through the VR app.

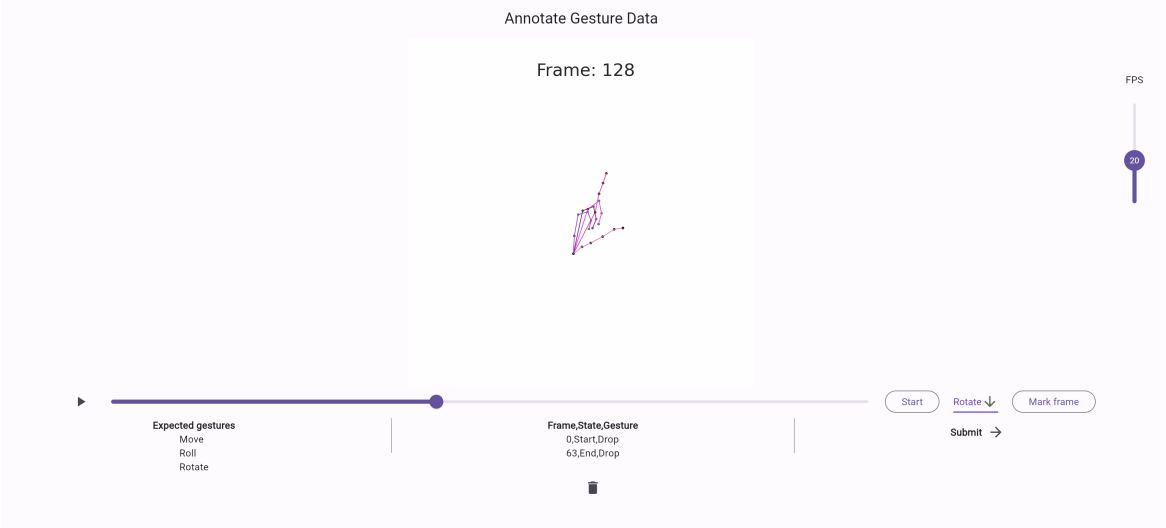


Fig. 4.3 Screenshot of web-based annotation tool. Expected gestures are indicated on the bottom left, the purple scrub bar is used to navigate the gesture frames, and the buttons on the right are used to label the start and end frames of a gesture.

11 participants provided hand gesture data. All users performed each gesture one to five times and a total of 30 gestures[1]. Table 4.1 shows the demographic information for all participants.

4.6 Recognition performance

In this section, I describe how the dataset was preprocessed, training a baseline model, leveraging a state-of-the-art spatial-temporal classifier, and changes made to improve the model.

4.6.1 Preprocessing

I load the data locally from the server, combining all of the CSVs for individual samples into one CSV file containing all labels. For every sample, there is a participant ID, dominant hand of the participant, sample UUID, gesture name, and start and end frame. Figure 4.4 shows the count of every gesture. As expected there are more roll and flip examples compared to other gestures, because users perform these with both hands rather than one. The gesture frequency imbalance is due to a random selection of gestures for each three gesture sequence in the VR data collection app. In future work, I would have all users perform the same number of each

[1]The number of times a gesture was repeated varied due to random selection of next gesture

Table 4.1 Participant demographics for data collection user study broken down by sex, age, dominant hand, and VR experience.

Category	Count
Sex	
Male	4
Female	7
Age	
18-25	7
26-35	1
35+	3
Dominant Hand	
Left	2
Right	9
VR Experience	
Yes	4
No	7

gesture and randomize the order in which they appear. Furthermore, I would consider asking participants to perform all gestures with both hands to gather more data without extending the data collection time per participant.

To conduct online inference to know which gesture a user is performing in the final study, I need to choose a fixed number of frames to include in a single sample. Figure 4.5 shows a plot of gesture lengths for all gestures. The selected fixed gesture length should favor buffering over cropping. Buffering refers to symmetrically repeating the first and last frame for each sample until the number of frames reaches the desired, fixed gesture length. Cropping refers to removing the beginning and end, symmetrically, of a gesture recording that exceeds the fixed gesture length. From examining Figure 4.5, I chose a fixed gesture length of 100, which captures the complete duration of 83.02% of samples. Figure 4.6 shows the buffering and cropping procedure in more detail. All model inputs are of dimension $S \times T$, where S corresponds to the number of samples in the batch, T corresponds to the number of frames or time steps, B corresponds to the number of bone IDs, and F corresponds to the features for each bone. Every feature vector is composed of a three-dimensional position vector concatenated with a four-dimensional quaternion vector. Quaternions are used to measure the rotation of each joint. Figure 4.7 visualizes the incoming tensor structure.

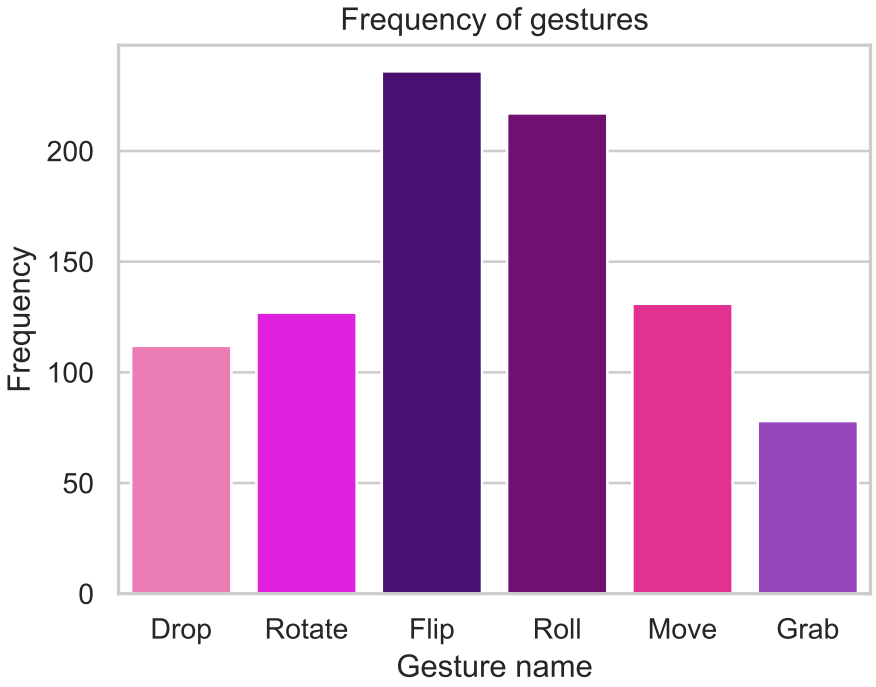


Fig. 4.4 Gesture frequency. The number of null gestures is dependent on the fixed gesture length.

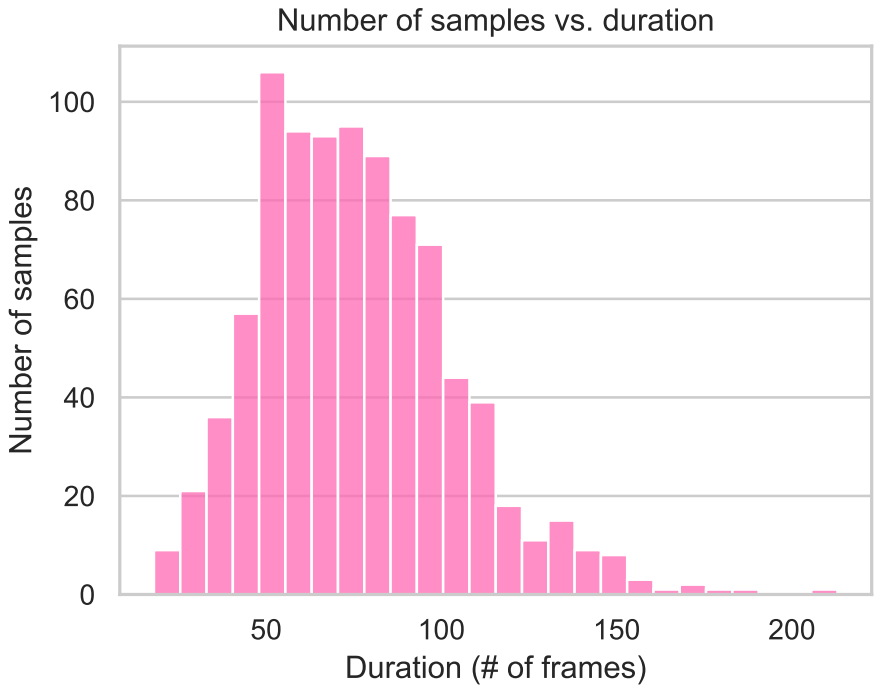


Fig. 4.5 Histogram showing the number of samples vs. the number of frames in each of the six main gestures. The plot is right-skewed, because the gesture must have a length of greater than 0 frames and some participants took a long time to complete gestures.

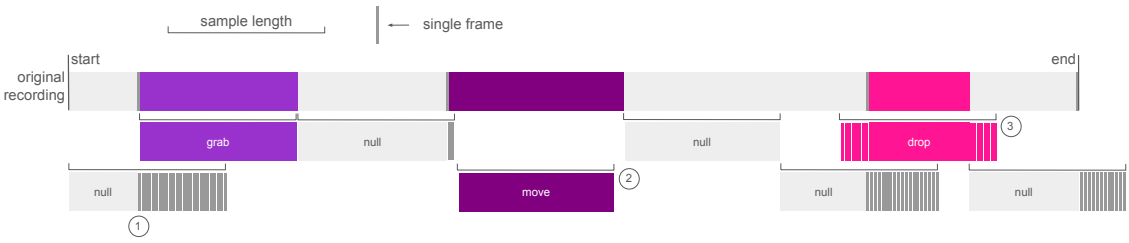


Fig. 4.6 Null gesture buffering logic. 1) For null samples that are less than the target sample length, we simply repeat the last frame until we reach the desired length. 2) For any of the six main gestures that last longer than the desired sample length, we symmetrically crop the beginning and end. 3) For main gestures shorter than the desired sample length, we repeat the initial frame and final frame, symmetrically, until the sample is of the target sample length.

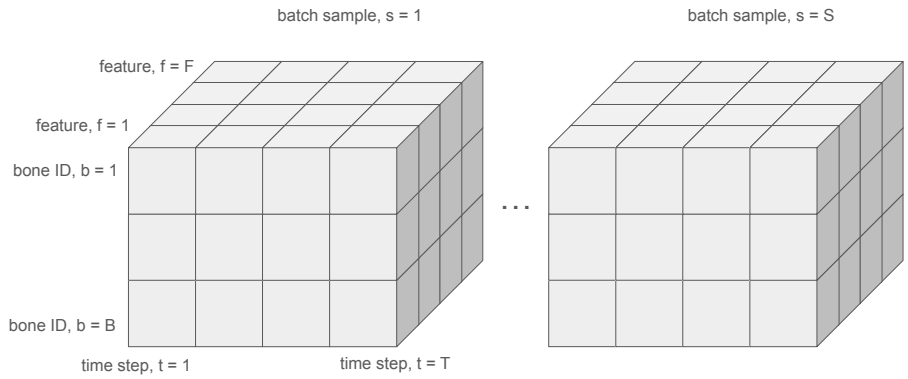


Fig. 4.7 The incoming data is four-dimensional: `batch` × `time` × `bone` × `feature`, where `batch` corresponds to the S samples in the current batch, `time` refers to the T frames in our sample, `bone` corresponds to the B bones, or joints, in a single time step, and `feature` specifies the F features for a single bone.

4.6.2 Baseline Model

I start with a basic Multilayer Perceptron (MLP) model, a simple model to benchmark subsequent results. For example, Wodziński and Krzyżanowska (2017) used the simple A* graph algorithm and an MLP to classify gesture data recorded with a Leap Motion sensor, which detects absolute joint positions. With minimal preprocessing, they achieved over 90% accuracy on 15 complex gestures (Wodziński and Krzyżanowska, 2017). I trained an MLP with the model structure shown in Figure 4.8 for 100 epochs. The average accuracy across three seeded runs was 38.50 ± 1.04. For the best performing seed of 0, Figure 4.10 shows the train and test curves, and Figure 4.9 shows the confusion matrix, normalized along columns to account for data imbalance. As expected, both Figures show that the model learned to predict the majority class, null gestures. An improved model should have a confusion matrix with a diagonal close to 1 and off-diagonal entries close to 0, unlike Figure 4.9.

Instead of continuing to improve this model, I implemented a state-of-the-art model for hand gesture recognition, STANet, which is a spatial-temporal, transformer-based model.

4.6.3 Transformers

Edwards and Xie (2017) first proposed skeleton graphs: representing the human skeleton as nodes connected via edges. In their setup, each vertex corresponds to a 3D point tracked with Motion Capture and the edges constitute bones (Edwards and Xie, 2017). Edwards and Xie (2017)'s best performing method was a Graph Convolutional Neural Network (GCNN), which improved on the baseline accuracy by over 15% at classifying human actions. A natural extension of this method would be to conceptualize the hand data recorded by the VR headset

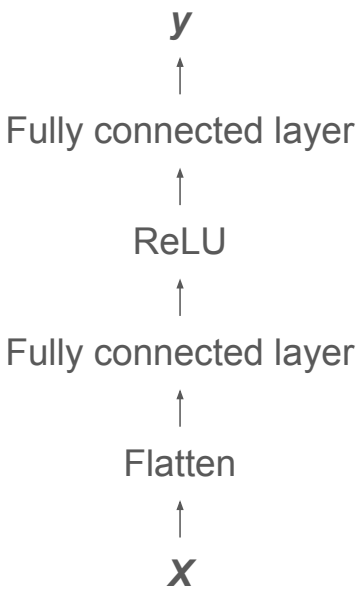


Fig. 4.8 MLP architecture. The input X is flattened from `batch` $\times$ `time` $\times$ `bone` $\times$ `feature` to `batch` $\times$ `features`, with $T \cdot B$ features.

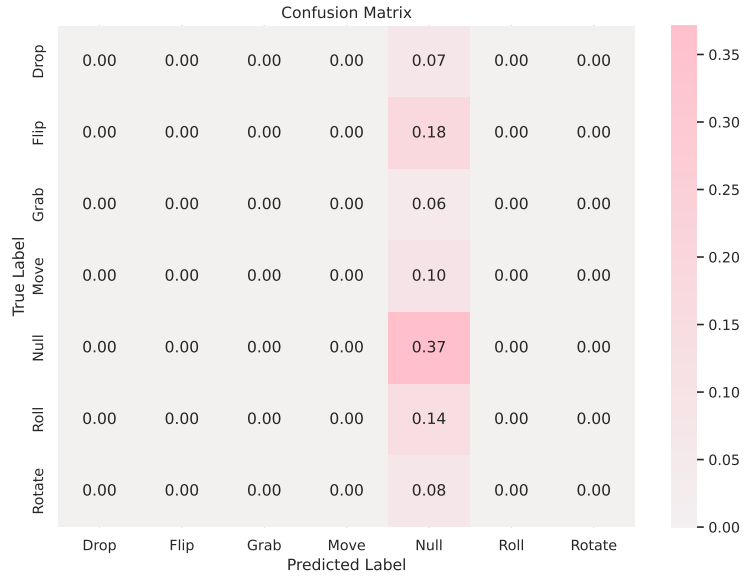


Fig. 4.9 Confusion matrix for trained MLP. Frequencies are normalized along columns. Note that the classifier only guesses the majority class null, so the fifth column is the only one with non-zero values. In contrast, for a “good" classifier, the diagonal should be close to 1 and the off-diagonal entries should be close to 0.

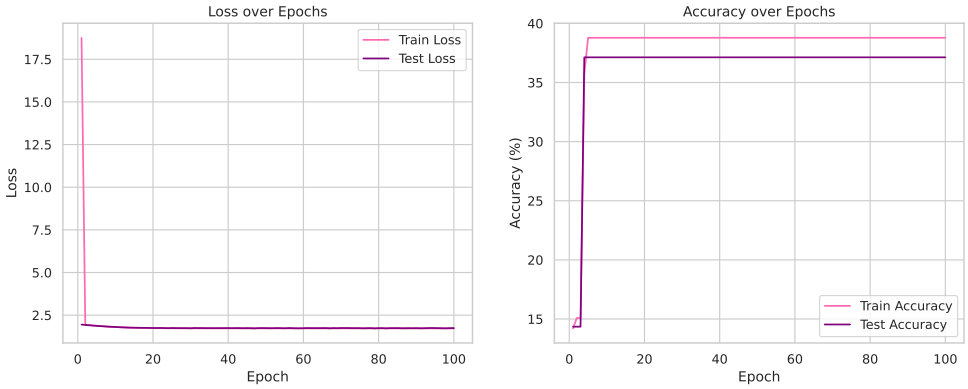


Fig. 4.10 Learning curves for the MLP, my baseline ML model. Just like the confusion matrix, the model learns to guess the majority class, and the model structure is too simple to learn more complex classification rules.

as graph structures, where nodes are joints and edges are bones. One common criticism of GCNNs is that they struggle to propagate information between distantly connected nodes.

Vaswani et al. (2017) propose the transformer, a model that learns via attention. When applied to skeleton data inputs, transformers learn the relationships between joints in the context of each gesture, rather than relying on the underlying bone structure. An example I gave in my poster presentation was try to bend your pinky finger while keeping your ring finger straight. Many peoples' ring finger bends in response to bending their pinky finger, illustrating long-range connections between nodes (joints), and motivating applying a transformer model to this problem.

In a transformer setup for a hand skeleton at a single time step, we consider n joints each with d features. In the case of using absolute position data, we have a three-dimensional position vector concatenated with a four-dimensional rotation vector, composed of quaternions. We arrange the n-column vectors of length $d = 7$ in a matrix; order does not matter because the model will learn the importance of how joint positions affect each other. Then, we perform self-attention. The following explanation of the transformer model is closely based on the setup presented by Turner (2024).

Let m be the number of layers in the transformer. Consider an n by n attention matrix $\mathbf{A}^{(m)}$. We compute the dot product of every row of the input matrix $X^{(m)}$, our d by n input matrix, with columns of $\mathbf{A}^{(m)}$, e.g.,

$$\mathbf{y}_n^{(m)} = \sum_{n'=1}^{N} \mathbf{x}_{n'}^{(m-1)} \mathbf{A}_{n',n}^{(m)}$$

where $\mathbf{x}_{n'}^{(m-1)}$ is the dth row and n'th column element of $\mathbf{X}^{(\mathbf{m})}$.

During training, the model learns to set high values for $\mathbf{A}^{(m)}_{n',n}$ if joints positions n and n' are dependent, and values close or equal to 0 if joint n is not influenced by the position and rotation of joint n'. The next section goes into detail about one method to compute $\mathbf{A}_{n',n}$.

One possible extension of this self-attention is multi-headed self-attention (MHSA). In MHSA, we have H matrices that represent the similarity between nodes, denoted $\mathbf{A}^{(}_{n}m)$. Viewed as full matrices, this becomes:

$$\mathbf{Y}^{(m)} = \text{MHSA}_\theta\left(\mathbf{X}^{(m-1)}\right) = \sum_{h=1}^{H} \mathbf{V}^{(m)}_h \mathbf{X}^{(m-1)} \mathbf{A}^{(m)}_h$$

where for each h, $\mathbf{A}^{(m)}_h$ can be unique. θ parameterizes the MHSA. We use MHSA instead of simple self-attention to allow joints to be similar in some dimensions and different in others, yielding a more flexible model.

After calculating $\mathbf{Y}^{(m)}$, we apply a non-linear transformation using an MLP, just like the one explored in the previous section (Section 4.6.2):

$$\mathbf{X}^{(m)} = MLP_\theta(\mathbf{Y}^{(m)})$$

where θ represents the parameters of the MLP and are constant for all n.

The final transformer is a combination of the self-attention step, MLPs, and normalization techniques. The exact architecture depends on the application and available computation power. I explore one possible setup in the next section.

4.6.4 STANet

Chen et al. (2019) apply the transformer architecture to the gesture recognition task, proposing a dynamic graph-based spatial-temporal attention (DG-STA) method that improves on traditional graph convolutional networks, tested on the SHREC'17 dataset, discussed earlier in Section 4.4 (Smedt et al., 2017). In this work and in the associated codebase, I refer to this model as a spatial-temporal attention network (STANet)[2], and describe the implementation here.

Let V be a fully connected skeleton graph consisting of nodes $v_{t,i}$, where $1 \leq t \leq T$ is the current time step up to time T, and $1 \leq i \leq N$ corresponds to the joints, up to N total joints. All nodes are connected to all other nodes including self-loops. Let us consider three types of edges:

[2]To put STANet in the context of Graph Neural Networks, the method is a variation of message passing, performed on a fully connected graph (Gilmer et al., 2017; Turner, 2024), where self-attention corresponds to the message passing step and the MLP corresponds to the feature update step.

1. spatial: edges at the same time step (except self-loops),

2. temporal: edges between different times steps[3], and

3. self-loops.

Chen et al. (2019) leverages two transformers, first a spatial attention model and then a temporal attention model. This is accomplished by first setting all temporal edges to zero, passing through the spatial transformer, and then setting spatial edges to zero to appropriately “mask" out the edges of no interest in each stage. Positional encoding is a simple transformation using sinusoidal functions to encode position in a sequence.

Combining with the explanation in the previous section (Section 4.6.3), let $\mathbf{x}_n$ correspond to the feature vector for node $v_{t,i}$. Chen et al. (2019) compute each element of the self-attention matrix $\mathbf{A}^{(m)}_{n',n}$ as:

$$\mathbf{A}^{(m)}_{n',n} = \frac{\exp\left((\mathbf{x}^{(m)}_{n'})^\top \mathbf{U}_\mathbf{k}^\top \mathbf{U}_\mathbf{q} \mathbf{x}^{(m)}_n / \sqrt{d}\right)}{\sum_{n''=1}^{N} \exp\left((\mathbf{x}^{(m)}_{n'})^\top \mathbf{U}_\mathbf{k}^\top \mathbf{U}_\mathbf{q} \mathbf{x}^{(m)}_n / \sqrt{d}\right)}$$

where $\mathbf{U}_\mathbf{k}^\top$ and $\mathbf{U}_\mathbf{q}$ are the only tunable parameters. $\mathbf{U}_\mathbf{k}$ and $\mathbf{U}_\mathbf{q}$ are matrices of shape $K \times d$, where typically $K < d$, because we only consider some features to compute the similarity between $\mathbf{x}^{(m)}_{n'}$ and $\mathbf{x}^{(m)}_{n'}$. In the transformer literature, the $\mathbf{k}$ stands for key and $\mathbf{q}$ for query, as a way to differentiate between the current node of interest n and all other nodes n' (Vaswani et al., 2017). By allowing $\mathbf{U}_\mathbf{k}^\top \neq \mathbf{U}_\mathbf{q}$, we permit asymmetric relationships between joints. Going back to our pinky example, if certain people put their pinkies down, then their ring fingers will bend too. But if they bend their ring finger, their pinky may or may not be bent. Lastly, we leverage a softmax function to compute the similarity of the input vectors for n and n' and divide by the square root of the number of features d to promote numerical stability.

Once the multi-headed attention has been computed for the spatial position encoding, the output is passed as input into the temporal positional encoding, the process of masking the spatial connections. Note however, that the input implicitly contains spatial information as it is the output of the spatial MHSA component. Then, pass the temporal position encoding through a temporal multi-headed attention layer. Normalize the output using layer normalization (LayerNorm). For LayerNorm, we normalize along each feature vector for every element of our batch (Turner, 2024). These steps taken together form the STABlock, shown in full in Figure 4.11.

[3]These edges may or may not connect the same node from time step t to time step k, where $t \neq k$.

Table 4.2 Varying dropout and number of epochs for STANet. Accuracies averaged over three seeded run results and reported as percentages. The best performing model was trained with a dropout of 0.3 for 100 epochs, and is **bolded** in the table.

	Number of epochs		
Dropout	50	100	150
0.1	93.55 ± 0.39	95.29 ± 1.02	94.15 ± 0.92
0.2	92.18 ± 0.57	94.99 ± 0.81	95.06 ± 1.32
0.3	91.50 ± 1.51	**95.75 ± 1.02**	93.70 ± 1.02

Song et al. (2023) improves on the original STANet model proposed by Chen et al. (2019), by addition additional linear layers and regularization such as layer normalization and dropout. Furthermore, I also add residual connections to mimic the setup by Vaswani et al. (2017). Figure 4.11 shows the salient parts of the STANet architecture, namely the half of the architecture for STABlock and the MLP setup used in STABlock. STABlock is the name of the model component that consists of, at a high level, the positional encoding for the spatial input, spatial MHSA, an MLP, the positional encoding for the temporal input, temporal MHSA, and a second MLP. The added residual connections are shown by the pink arrows.

4.6.5 Training results for STANet with absolute positions

Table 4.2 shows the results of tuning the number of epochs and dropout of the STANet model. The best performing model was trained with a dropout of 0.3 over 100 epochs. The results in Table 4.2 are computed from 3 seeds. The best performing seed under this condition was a seed of 2 with 97.03% test accuracy. The corresponding training curve is shown in Figure 4.13 and the confusion matrix across all 7 classes is shown in Figure 4.12.

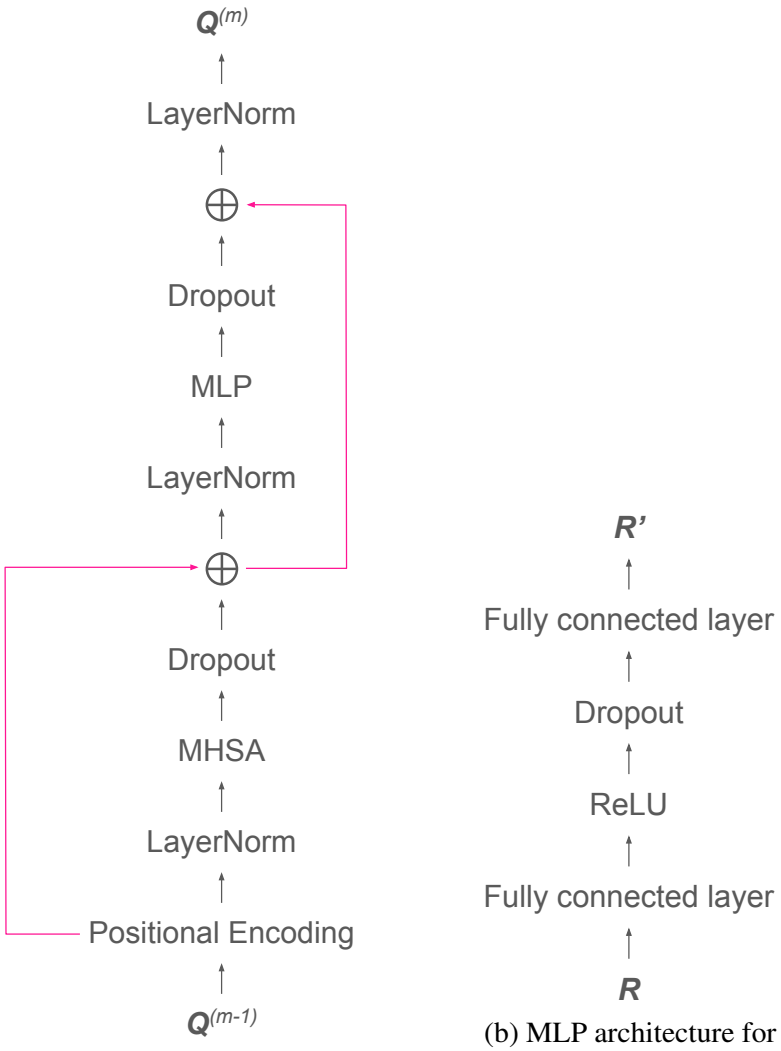
4.6.6 Online inference

I initially tested online inference using my own hands and the model from the previous section (Section 4.6.5). I accepted up to 100 frames from the headset and then made a prediction, using a buffer and asynchronous execution to not block the main thread accepting new data. Another student and I confirmed that the model outputs made no sense, even though inference with test samples was achieving over 95% accuracy. To improve the model, I first computed the relative positions between every other bone and the wrist, indicated in Figure 4.14.

Furthermore, I replaced the wrist feature data with each wrist relative to the first wrist ($t = 1$) in each sample, indicated in Figure 4.15. This slightly improved the model when testing with my own hands.

Next, I followed the work of Song et al. (2023) and reduced the number of frames per sample from 100 to 50, and again saw a minor qualitative improvement. Also like Song et al. (2023), I re-trained my model to use first the left hand samples and then the right hand samples, and saw further improvement. Finally, I considered using wrist velocity instead of the more extreme value-producing wrist relative to first wrist method. Figure 4.16 shows a pictorial representation of the wrist velocity calculation. By using the velocity, I capture the direction of motion for each gesture, allowing the model to differentiate between gestures like Drop and Roll that have similar hand shapes (i.e., thumb extended or curled) but different directions of movement.

The final model was trained over 50 epochs with a learning rate of $10\exp -3$. The model converged in fewer epochs using the relative inputs compared to the absolute inputs.



(a) Half of STABlock.

(b) MLP architecture for STANet.

Fig. 4.11 Components of STANet architecture. The spatial information is passed through half of the STABlock (shown in Figure 4.11a) and then the temporal entries come into the second half of the STABlock. This architecture contains the two essential components of a transformer: attention (MHSA) and a non-linear transformation (MLP). The MLP is expanded upon in Figure 4.11b, and is similar to the architecture of the baseline model, shown in Figure 4.8.

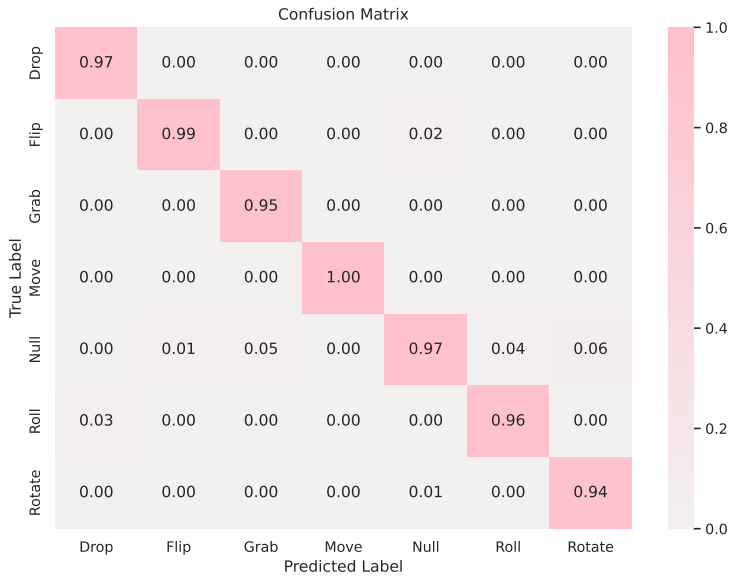


Fig. 4.12 Confusion matrix STANet. Frequencies are normalized along columns. As desired, the off-diagonals are close to 0 and the diagonal entries are close to one.

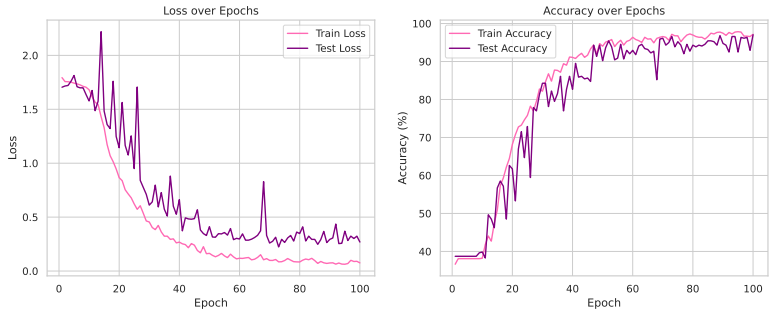


Fig. 4.13 Learning curves for STANet for best performing model trained for 100 epochs, dropout of 0.3, and a seed of 2.

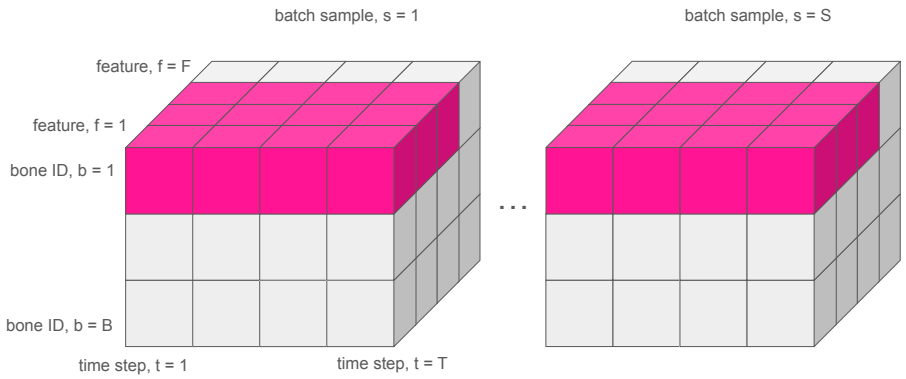


Fig. 4.14 Data structure highlighting wrist positions in pink. I only compute relative positions for the position coordinates, not the quaternions, so only the front half of the wrist features are colored pink.

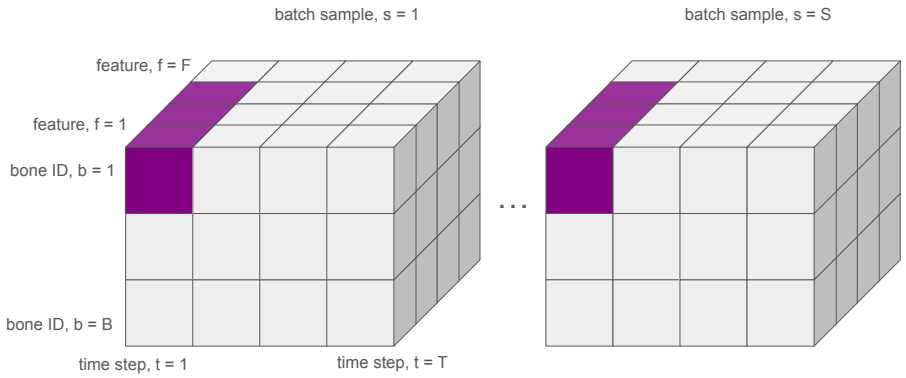


Fig. 4.15 Data structure highlighting wrist at time step $t = 1$. In order to compute the velocity of the wrist, I must select the first wrist. This feature vector is repeated to compute a difference.

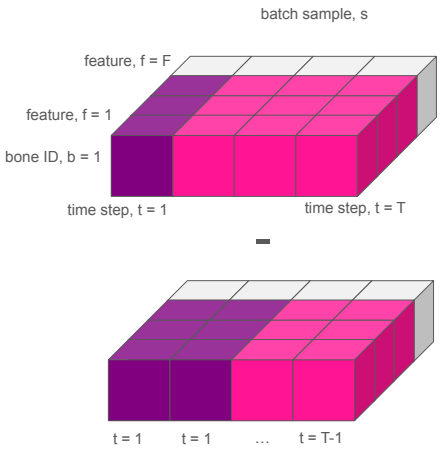


Fig. 4.16 Data structures for wrist velocity calculation. The feature vectors for the wrists across all time steps are computed. Using the difference shown pictorially here, the initial velocity would equal 0, so we choose to set the wrist velocity at time step 1 equal to the wrist velocity at time step 2 to avoid a large acceleration from the first to the second time step.

Chapter 5

User evaluation

5.1 Overview

In this chapter, I consider a user study and think-aloud study results. Participants completed a pick-and-place tasks using two robot arms, controlled via abstracted control and direct control. They were asked to compare the method of selecting objects with their eyes and indicating actions with gestures to a method using haptic input controllers. The goal of the study was to compare efficiency, perceived mental load, and safety of both cases. In this chapter, I examine dictated feedback from the user study, collected through a series of think-aloud questions asked during and after the study to probe users' perceptions of both the direct and abstracted control cases.

5.2 Study design

I consider two conditions in my user study: direct control and abstracted control. I conduct a within user study; all participants attempt both the direct and abstracted control conditions. I randomly assign users to the order in which they encounter each condition. For each participant, I place three blocks and three goal positions, each within the bounds of the arms. For all participants, two blocks were accessible to one arm and one block was accessible to the other arm. Aided by the external camera providing real time feedback, the positions and blocks could be rearranged at the start of each iteration of the task. The robot arms always started at the back of the set up, out of the way of the external camera mounted overhead tracking the objects in the scene. Users also completed the gaze tracking calibration exercise after putting on the VR headset to ensure proper gaze-tracking throughout abstracted control portion of the study.

During the study, I asked the 3 participants to consider the following questions:

1. How easy is it to use this form of control?
2. What strategy are you using to control two robot arms?
3. How do you feel the system is doing at performing the actions you want?
4. Did you/would you control both arms at the same time?

The purpose of the third question is to understand how the system is interpreting user intent, and the forth question aims to address the question of whether abstracted control makes it easier to control multiple arms at once. Some provided their thinking during the exercise and others gave detailed answers at the end.

5.3 Task description

The physical and simulated set ups used in the user study are shown in Figure 3.1. For each condition, I place 3 interactable objects (blocks) and 3 interactable positions on the grid[1]. Using the rules established in Section 3.5, the block and positions are colored in the virtual scene to indicate which elements go together. Users wear the VR headset and are instructed to place each colored block on the goal position with the same color. The picking and placing action is dependent on the condition. In the abstracted case, users looked at the object or position that they wanted to interact with. The element changed color after the user gaze at it for at least 0.5 seconds and changed from a dull to brighter version of the same color to indicate selection. Users could select up to one interactable object and one interactable position at a time to ease the system's discernment of user intent. If a user placed a block touching the same color position (even just minimal overlap), it was considered successfully placed. The video linked here shows short snippets from my demonstration of each condition and a participant completing the user study[2].

5.4 Participants

Participant sampling was done via convenience sampling of adults associated with the University of Cambridge. Three participants were recruited to complete the study, and from

[1]The grid was not actually used in the system design; it was merely an artifact leftover from a previous researcher who used the setup.

[2]The participant provided explicit consent to be included in a demonstration video for this piece of work.

Table 5.1 Participant demographics for user study broken down by sex, age, dominant hand, and VR experience.

Category	Count
Sex	
Male	2
Female	1
Age	
18-25	3
26-35	0
35+	0
Dominant Hand	
Left	0
Right	3
VR Experience	
Yes	2
No	1

start to finish each participant spent about 45 minutes interacting with the system. This consisted of the time to

- adjust the headset,
- calibrate gaze-tracking,
- get comfortable wearing the headset and moving hands in VR,
- practice maneuvering the haptic input devices,
- test out the controls for each condition,
- perform the task for each condition, and
- provide additional feedback on the task.

Table 5.1 shows the demographic data for the user study.

5.5 Results

I analyze responses to the think-aloud questions here. I focus on participant feedback related to the two methods of control, including the ease of controlling two arms at once, the gaze-tracking selection and gesture performance, mental strain, and safety. I briefly touch on system design feedback common to both conditions.

5.5.1 Ease of use

All users agreed that the abstracted control case was simple enough. Participant 1 (P1) claimed that the “gestures are easy to do" and Participant 2 (P2) said “once I got the hang of it, it’s pretty easy." All users had the opportunity to practice the controls for both cases until they felt comfortable.

For the direct control case, users found the process “relatively straightforward." One user described it as “it is practically very easy and intuitive, because my motions are translated directly to things (the robot arms) that I can understand in the scene. I move right, it moves right." Since users may be familiar with handheld controllers, this one-to-one correspondence between controller and arm was simple for users to grasp quickly. However, one participant achieved the same one-to-one correspondence in the abstracted control case by using the “left hand for the left arm" and “treat[ing] the set up like a mirror."

All users were able to complete the task under both conditions. Two users moved only one block at a time in both conditions, choosing to watch the blocks until they were in the correct position in the abstracted control case. P3 managed to pick up two blocks at the same time and place two blocks at the same time, leveraging both arms in the abstracted control case. All users were able to indicate the object they wanted to pick up with more ease in the abstracted case compared to the direct case, with Participant 3 (P3), exclaiming, “it’s finicky!" during the direct control scenario.

5.5.2 Gaze-tracking to select objects

Two participants talked about the eye-tracking to make a selection. P2 stated, “[using my] eyes is much easier than the other case" and “when it’s highlighted [by my eyes], I do the gesture." There were some minor complaints that the “eye-tracking was a little jumpy" (P1) and “the selection feature could be a bit better. If I gaze at it (the interactable object or position), I should not have to move to get to a better angle" (P3). P3 was referring to how the gaze tracker logic returns the object that the ray cast extending from the eyes intersects with first (. For some initial set ups, e.g., the interactable positions are behind the interactable

objects, participants had to lean over the blocks to make the desired selection (as shown previously in Figure 3.9).

5.5.3 Controlling two arms

The main drawback of the direct case was that users only manipulated one arm at a time. P1 said it was “more difficult because I had to hold two things at the same time [rather] than just doing gestures." During the study, I observed that P1 also seemed to forget about the arm that they were not focused on using, and moved that forgotten controller around aimlessly. P2 said in response to being asked about controlling two arms at once that they “didn’t try this" in the direct control case because they had “a lot of attention on one block." P3 separately corroborated this point stating that in the direct control case, they “did not think about doing it with two hands" and, in contrast, the abstracted case “frees up my mental capacity to do (control) both [arms] at the same time." Furthermore, P3 stated that they “don’t have to actively participant in the moving....obviously there is a lag as the system carries it out, but I’m not actively involved in the process."

Overall, participants reported that they were more likely to control two arms at once in the abstracted control case compared to the direct control case. They liked the eye-tracking feature to select the exact interactable object or position of interest. While they all found the haptic input device controllers in the direct control case intuitive, they felt like the abstracted case required less myopic focus.

5.5.4 Lack of real-time feedback

In this chapter, I focus on user evaluation specific to each condition. However, one common complaint for both conditions from all users was that the block positions did not update in real time. In the abstracted case, users had to place the block and wait for the arm to move back to the reset position for the external camera to update the VR scene to match the real scene. In the direct control case, users had to manually move the arms out of the way to see the updated block positions. For example, P2 stated that there was “some issue with correspondence between the VR scene and the real scene."

5.6 Analysis

The participant statements regarding control of two arms at once are in line with what Warattaseth et al. (2024) hypothesized; with abstracted control, users would be able to take advantage of the latent time when the robot is executing a behavior perform actions with

the other arms. This could accelerate task completion time and reduce mental strain. P3 demonstrated parallel actuation, using both arms at once to pick up blocks. Though they chose to focus on just one block at a time, P1 and P2 were also able to pick up objects more easily in the abstracted control case, due to the precision of the gaze-tracking.

While some participants complained about the gaze-tracking implementation, in particular having to move to select object towards the back of the scene, this is at the very least possible because of the nature of a three-dimensional scene in VR. With a camera set up only, teleoperators would need to change the orientation of the camera to access hidden objects, rather than just adjusting their body posture.

The system failure of not updating the position of objects in real time until the robot arms were moved out of the way of the camera impacted the direct control case much more than the abstracted control case. In the abstracted control case, the system handles the precise positioning of the robot arms by allowing users to select the exact center of scene elements with their gaze. It only matters that users see the updated positions to ensure they placed the block on the correct colored target position. Consequently, the abstracted case requires additional logic set up to establish which elements of the scene are interactable objects and positions.

Finally, without proper controls in place, P1 aimlessly moving the haptic input device that they were not focused on could have resulted in dangerous operation. Fortunately, in the system design I set a maximum speed and placed spatial limits on the robot arms (i.e., the arms can only reach positions in a fixed cube, separate from the other robot's spatial cube). This could also make the abstracted control case a safer method of control, because users are less likely to hyper-focus on one robot arm, reducing the likelihood of potentially dangerous manipulation of the other controller.

Abstracted control is a promising improvement over direct control. In general, users found it easier to select objects, had the option to move both arms at the same time, and reported needing to focus less intensely on a single object. Given that abstracted control case is machine intelligence-assisted, it could be a safer approach.

Chapter 6

Discussion & conclusion

6.1 Research questions revisited

In this work I addressed the research questions introduced in Section 1.3. From the user evaluation, I found that the abstracted method of control was easier for users to indicate intent and allowed some to actuate both robot arms at the same time. All participants were reasonably comfortable using gaze-tracking for object selection and hand gestures for action selection. Quantitative results like task completion time could further support these findings.

6.2 Data collection limitations

In future iterations of the data collection app, I would split up the JSON data containing hand-tracking data automatically into smaller segments of no more than 600 frames for faster post-processing. Furthermore, I would add conditional logic to the annotation tool, such as enforcing that a gesture start frame precedes the gesture end frame. When loading my data for training, I had to go back and edit some of the labels by hand because I mistakenly put the end frame before the start frame.

Due to time constraints, I was the only data annotator. To minimize bias around when a gesture starts and ends, I would have at least one other person label the data and then average the start frames and end frames. If there was total mismatch, e.g., one person labeled a gesture differently than the other annotator, I would have a programmatic check in place to flag the mismatch for re-review. In general, having multiple annotators and inter-annotator agreement improves dataset quality for machine learning tasks(Pfitzmann et al., 2022).

Finally, I would randomize the order of gestures that the user sees, but all users would perform all gestures with both hands an equal number of times to produce a balanced dataset.

6.3 Modelling limitations

In my pre-processing step, I buffer positive gesture samples that are too short (Figure 4.6). For some gestures, the online gesture recognizer outputs the gesture from the static hand shape rather than the hand shape with the movement. To address this issue, I would add the buffering frames as null gestures so the model learns to not recognize still hands as gestures.

6.4 System design limitations

The external camera was mounted directly above the scene and captured movement in the x and y dimensions (refer back to Figure 3.8), hence no real time data was shown indicating whether a block had been picked up, until a user dropped the block and moved the arms out of the way. While I considered mounting the single external camera at an angle to capture the z-dimension, I was already experiencing distortion from just the overhead view and the additional complexity of a third dimension would have likely caused even more calibration issues, without a substantial amount of time spent tuning the transformation from camera space to the universal coordinate system of the left robot arm. With a second external camera, I would be able to track blocks in the z-dimension, using the same logic as before. I would also need to mark objects on all sides with April tags and match up the feeds from the two cameras. The RealSense camera that I used does have depth sensing, which, if mounted and calibrated properly, could allow for sensing objects of different sizes (all of the blocks I used were the same width, length, and height), and tracking the z position.

For the abstracted control case, the interactable objects and positions must be pre-specified so that the Unity knows which objects are interactable. It would be better to have a more flexible method of specifying which objects can be manipulated by the robot arms.

6.5 User study limitations

Due to time, safety, and computational constraints, certain simplifications had to be made to the user study. First, I only considered a pick and place task. This prevented users from needed to rotate the end effector; in the direct control case this can lead to dangerous behavior without extensive additional robotics control. For instance, in the simulated version, the robot arms often strike the table, attempting to reach certain rotated positions. Thus, I chose a task that kept the robot aligned in the z-axis direction throughout.

Next, due to time constraints, I was unable to simulate the exact position of blocks when the robot arm was above it. This deficiency was highlighted by all participants in the final

user study, with one participant stating, “It was okay besides the calibration of whether I was actually picking [the block] up in real life." To provide real time feedback, I answered users when they asked me whether they had picked up the block or not in the direct control case. To get around this issue during the user study, I instructed users to move the arms out of the way in the direct control case so that the external camera could update the position of the objects in the scene. While this added some time and frustration, it made it possible for users to have only a slightly delayed perception of the real scene.

In the study, users could see their hands performing the gestures in the abstracted control case, but I manually called the gestures on the server-side. This was because the server could not respond to the requests, given all of the other processing that needed to occur to control the robot arms, accept input from the external camera, and communicate updates with many elements of the Unity scene. To make the gesture recognizer work, I would need to run it on a separate server and then communicate the results to the main server, which I did not have time to implement. While disappointing to not be able to use the gesture recognizer in the user study, it was also safer to ensure I had control over when the physical robots executed the movements.

6.6 Future works

6.6.1 System setup improvements and additional evaluation

Before using this system in future work, I would improve the system by addressing the system design and user study limitations. In particular, I would update the camera setup so that objects are updated in the Unity scene in real time, not with a slight delay. Furthermore, I would put the gesture recognizer on a separate server to reduce the strain on the existing server.

With these changes in place, I would repeat the pick-and-place task with more participants and trials per participant (i.e., reset the blocks and repeat moving them to target positions). I would also time each trial to produce quantitative results comparing abstracted and direct control.

6.6.2 Additional tasks beyond pick-and-place

I originally designed three tasks to illustrate that this setup can use abstracted control to

- ease operation of multiple robots at the same time, and
- leverage additional robots with less mental strain,

Table 6.1 Descriptions of additional tasks, highlighting the necessary gestures and purpose of gaze-tracking to implement abstracted control. For all tasks direct control would remain the same, i.e., users can move controllers, rotate the end effector, and turn the grabber on or off.

	Pick-and-place	Roll pizza dough	Build table
Why this task?	parallel actuation	requires multiple arms	parallel actuation and multiple arms
Gestures	grab, drop	grab, move, drop, roll, rotate	grab, move, drop, rotate, flip
Gaze-tracking	discriminate between objects	choose where to roll next	discriminate between objects

compared to direct control.

In this work, I describe, implement, and evaluate the pick-and-place task with multiple robotic arms (Figure 6.1a). The purpose of this task was to show how operating multiple robot arms is easier for users by using gestures and gaze-tracking.

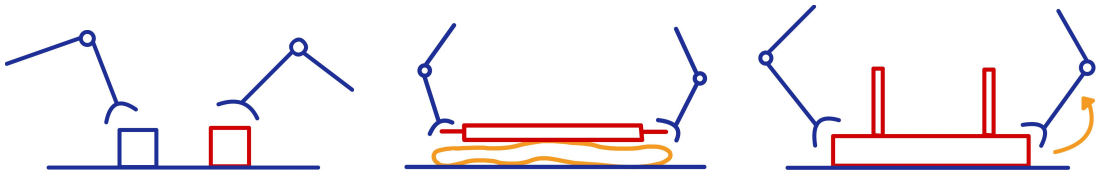
The second task that I would want to implement and evaluate is rolling out pizza dough (Figure 6.1b). While the pick-and-place task could technically be accomplished with a single robot arm, rolling pizza dough requires at least two arms to hold the rolling pin.

The third task would be building a table, which was implemented as a multi-arm collaborative teleoperation task by Tung et al. (2021). In this task (Figure 6.1c), I would ask users to place legs of a table into a base. Once all legs are in place, the user would need to use both robot arms at once to flip the table into an upright position. This task could benefit from both

- simplified parallel actuation to place the legs in the base more quickly, and
- multiple arms to flip the table at the end,

showcasing the two major proposed improvements of abstracted control over direct control.

Table 6.1 summarizes the purpose of each task and shows the gestures and gaze-tracking use for the abstracted control case.



(a) Pick-and-place task. (b) Roll pizza dough task. (c) Build table task.

Fig. 6.1 Diagrams for 3 tasks for a multi-arm system.

6.6.3 Applying Bayesian logic to gesture recognizer

Zhu et al. (2020) proposes a method called BayesianCommand to reduce noise and improve predictions by combining the likelihood and prior of some input data. They demonstrate

improved prediction on a point-and-click task. Bayes' Theorem states:

$$c^* = \text{argmax}_{c \in C} P(c|\mathbf{s}) = \text{argmax}_{c \in C} \frac{P(\mathbf{s}|c)P(c)}{P(\mathbf{s})}$$

where $P(\mathbf{s})$ is fixed for given input $\mathbf{s}$ and $c = \{c_1, c_2, ..., c_n\}$, which are possible predicted actions. $P(c)$ is the prior probability, and $P(\mathbf{s}|c)$ is the likelihood. This posterior probability can be used for Bayesian inference.

In my gesture recognition setup, the model output is the likelihood $P(\mathbf{s}|c)$, where each c_i for $1 \leq i \leq n$ is a gesture. The prior $P(c)$ could come from the frequency of a gesture being performed after the previously recognized gesture. For example, the prior could encode that if a user previously grabbed an object, they are likely to place it. The prior probability distribution is adjusted for each user in real-time as they interact with the system more. This method would allow users to have different task strategies. In the pick-and-place task evaluation presented in this work, participants 1 and 2 picked up an object and then placed it before picking up a second object. Participant 3 chose to pick up two objects and then place both objects. By updating the prior probability in real-time, a new method leveraging BayesianCommand could account for these two different participant strategies. I hypothesize that Bayesian gesture inference could improve online gesture recognition accuracy.

6.7 Conclusion

In this work, I present how abstracted control can ease parallel actuation of two robot arms in teleoperation. I use multi-modal input (gaze-tracking and hand-tracking) collected from a VR headset to infer what actions a user want to perform on which objects in a pick-and-place task. This effectively reduces consecutive two-step process of selecting an action and then an object to apply that action to to a a parallel process; users demonstrated that they could select an action to perform and an object to perform it on at the same time. In the user study evaluation, I compared this method of abstracted control to direct control and demonstrated how users could more easily control both arms at the same time in the abstracted control case. In order to support this abstracted control case, I curated a new gesture dataset and trained an attention-based model called STANet to recognize gestures in real time. In future work, I would demonstrate how the abstracted form of control can improve efficiency and reduce mental strain for additional tasks require multiple robots.

All code can be found in this private GitHub repository. Email ceh210@cam.ac.uk to request access.

References

Abdlkarim, D., Di Luca, M., Aves, P., Maaroufi, M., Yeo, S.-H., Miall, R. C., Holland, P., and Galea, J. M. (2023). A methodological framework to assess the accuracy of virtual reality hand-tracking systems: A case study with the meta quest 2. Behavior Research Methods.

Almeida, L., Menezes, P., and Dias, J. (2017). Improving robot teleoperation experience via immersive interfaces. In 2017 4th Experiment@International Conference (exp.at'17), pages 87–92.

Burdea, G. C. (1999). Invited review: the synergy between virtual reality and robotics. IEEE Transactions on Robotics and Automation, 15(3):400–410.

Chen, Y., Zhao, L., Peng, X., Yuan, J., and Metaxas, D. N. (2019). Construct dynamic graphs for hand gesture recognition via spatial-temporal attention.

Dudley, J. (2024). Mopteleop. Private repository.

Edwards, M. and Xie, X. (2017). Graph-based cnn for human action recognition from 3d pose. In British Machine Vision Conference Workshop: Deep Learning on Irregular Domains, pages 1.1–1.10.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. (2017). Neural message passing for quantum chemistry. In Proceedings of the 34th International Conference on Machine Learning - Volume 70, ICML'17, page 1263–1272. JMLR.org.

Hokayem, P. F. and Spong, M. W. (2006). Bilateral teleoperation: An historical survey. Automatica, 42(12):2035–2057.

Karam, M. and m. c. schraefel (2005). A taxonomy of gestures in human computer interactions. Project report, University of Southampton.

LI, Y., HUANG, J., TIAN, F., WANG, H.-A., and DAI, G.-Z. (2019). Gesture interaction in virtual reality. Virtual Reality & Intelligent Hardware, 1(1):84–112.

Ly, K. T., Poozhiyil, M., Pandya, H., Neumann, G., and Kucukyilmaz, A. (2021). Intent-aware predictive haptic guidance and its application to shared control teleoperation. In 2021 30th IEEE International Conference on Robot & Human Interactive Communication (RO-MAN), pages 565–572.

Masurovsky, A., Chojecki, P., Runde, D., Lafci, M., Przewozny, D., and Gaebler, M. (2020). Controller-free hand tracking for grab-and-place tasks in immersive virtual reality: Design elements and their empirical study. Multimodal Technologies and Interaction, 4(4).

Olson, E. (2011). AprilTag: A robust and flexible visual fiducial system. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), pages 3400–3407. IEEE.

Pace, F. D., Gorjup, G., Bai, H., Sanna, A., Liarokapis, M., and Billinghurst, M. (2021). Leveraging enhanced virtual reality methods and environments for efficient, intuitive, and immersive teleoperation of robots. In 2021 IEEE International Conference on Robotics and Automation (ICRA), pages 12967–12973.

Peppoloni, L., Brizzi, F., Ruffaldi, E., and Avizzano, C. A. (2015). Augmented reality-aided tele-presence system for robot manipulation in industrial manufacturing. In Proceedings of the 21st ACM Symposium on Virtual Reality Software and Technology, VRST '15, page 237–240, New York, NY, USA. Association for Computing Machinery.

Pfitzmann, B., Auer, C., Dolfi, M., Nassar, A. S., and Staar, P. (2022). Doclaynet: A large human-annotated dataset for document-layout segmentation. In Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD '22, page 3743–3751, New York, NY, USA. Association for Computing Machinery.

Sinha, R., Sanjay, M., Rupa, B., and Kumari, S. (2015). Robotic surgery in gynecology. Journal of Minimal Access Surgery, 11(1):50–59.

Smedt, Q. D., Wannous, H., Vandeborre, J.-P., Guerry, J., Saux, B. L., and Filliat, D. (2017). Shrec 2017 track: 3D hand gesture recognition using a depth and skeletal dataset. In 3DOR - 10th Eurographics Workshop on 3D Object Retrieval, pages 1–6.

Song, Z., Dudley, J. J., and Kristensson, P. O. (2023). Hotgestures: Complementing command selection and use with delimiter-free gesture-based shortcuts in virtual reality. IEEE Transactions on Visualization and Computer Graphics, 29(11):4600–4610.

Tung, A., Wong, J., Mandlekar, A., Martín-Martín, R., Zhu, Y., Fei-Fei, L., and Savarese, S. (2021). Learning multi-arm manipulation through collaborative teleoperation. In 2021 IEEE International Conference on Robotics and Automation (ICRA), pages 9212–9219.

Turner, R. E. (2024). An introduction to transformers.

Van de Merwe, D. B., Van Maanen, L., Ter Haar, F. B., Van Dijk, R. J. E., Hoeba, N., and Van der Stap, N. (2019). Human-robot interaction during virtual reality mediated teleoperation: How environment information affects spatial task performance and operator situation awareness. In Chen, J. Y. and Fragomeni, G., editors, Virtual, Augmented and Mixed Reality. Applications and Case Studies, pages 163–177, Cham. Springer International Publishing.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. (2017). Attention is all you need. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, Advances in Neural Information Processing Systems, volume 30. Curran Associates, Inc.

Wang, W., Chen, Y., Li, R., and Jia, Y. (2019). Learning and comfort in human–robot interaction: A review. Applied Sciences, 9(23):5152.

Warattaseth, P., Abdulali, A., Dudley, J., and Iida, F. (2024). Efficient robot control with motion primitives and mixed reality.

Wodziński, M. and Krzyżanowska, A. (2017). Sequential classification of palm gestures based on a* algorithm and mlp neural network for quadrocopter control. Metrology and Measurement Systems, 24(2):265–276. Received on Jun. 30, 2016; accepted on Oct. 22, 2016; available online on Jun. 30, 2017.

Wonsick, M. and Padir, T. (2020). A systematic review of virtual reality interfaces for controlling and interacting with robots. Applied Sciences, 10(24).

Xu, X., Gong, J., Brum, C., Liang, L., Suh, B., Gupta, K., Agarwal, Y., Lindsey, L., Kang, R., Shahsavari, B., Nguyen, T., Nieto, H., Hudson, S. E., Maalouf, C., Mousavi, S., and Laput, G. (2022). Enabling hand gesture customization on wrist-worn devices. In CHI.

Zhu, S., Kim, Y., Zheng, J., Luo, J. Y., Qin, R., Wang, L., Fan, X., Tian, F., and Bi, X. (2020). Using bayes' theorem for command input: Principle, models, and applications. In Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems, CHI '20, page 1–15, New York, NY, USA. Association for Computing Machinery.